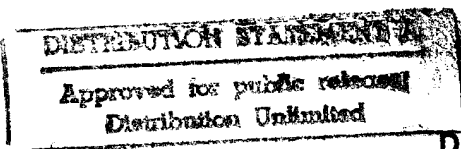


A ROBUST METHOD OF SOLVING NONLINEAR  
BOUNDARY VALUE PROBLEMS VIA MODIFIED  
COMPROMISE PROGRAMMING

THESIS

John L. Zornick, Captain, USA

AFIT/ENC/GOR/95J-01



DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

DTIC QUALITY INSPECTED 5

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

26/1

AFIT/ENC/GOR/95J-01

19950811 046

A ROBUST METHOD OF SOLVING NONLINEAR  
BOUNDARY VALUE PROBLEMS VIA MODIFIED  
COMPROMISE PROGRAMMING

THESIS

John L. Zornick, Captain, USA

AFIT/ENC/GOR/95J-01

Approved for public release; distribution unlimited

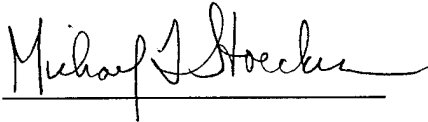

## THESIS APPROVAL

STUDENT: John L. Zornick  
Captain, USA

CLASS: GOR95J

THESIS TITLE: A ROBUST METHOD OF SOLVING NONLINEAR BOUNDARY  
VALUE PROBLEMS VIA MODIFIED COMPROMISE  
PROGRAMMING

DEFENSE DATE: 18 May 1995

COMMITTEE	NAME/DEPARTMENT	SIGNATURE
Advisor	Captain Michael Stoecker Assistant Professor, Department of Mathematics and Statistics	
Reader	Dr. Yupo Chan Professor, Department of Operational Sciences	

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

The views expressed in this thesis are those of the author and do not reflect the official policy or positions of the Department of Defense or the U. S. Government.

AFIT/ENC/GOR/95J-01

A ROBUST METHOD OF SOLVING NONLINEAR BOUNDARY VALUE  
PROBLEMS VIA MODIFIED COMPROMISE PROGRAMMING

THESIS

Presented to the Faculty of the Graduate School of Engineering  
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of  
Master of Science in Operations Research

John L. Zornick, B.S.

Captain, USA

MAY, 1995

Approved for public release; distribution unlimited

## Preface

The purpose of this research was to improve a goal programming technique that was developed to solve nonlinear boundary value problems. The goal programming method of solving such problems was developed by Kevin Ng of the Canadian Department of Defense. While Ng developed a very simple and novel approach to such problems that produced excellent results, it seemed that he left room for others to improve on his work. My interest in genetic algorithms is what prompted me to adopt the modification described in this document. While my initial intention was only to improve on the optimizing engine of Ng's basic model, as I learned more about genetic algorithms it became apparent that I could further simplify his technique. Hence, I arrived at what I have termed "modified" compromise programming.

This research would not have been possible without the help of others. I owe a great deal of gratitude to my research advisor, Captain Mike Stoecker. Not only did he keep me going in the right direction throughout, but his ability to see through complex problems led directly to the modification which I adopted ("but the book says to set it up this way!"). Additionally, Dr. Yupo Chan provided much insight into the field of compromise programming. I would also like to thank Dr. Gary Lamont for lending his expertise in the area of genetic algorithms. Lastly, I would like to thank someone who knows nothing about compromise programming, genetic algorithms, or boundary value problems for that matter -- my wife Pam. But she does know all about "supporting the mission" and that is what she did throughout my quest. If only I possessed her superb organizational abilities.

# Table of Contents

	Page
Preface.....	ii
List of Figures.....	vi
List of Tables.....	vii
Abstract.....	viii
I. Introduction	
Background.....	1-1
Problem.....	1-4
Scope.....	1-5
General Approach.....	1-6
Presentation.....	1-7
II. Preliminaries	
Introduction.....	2-1
Limitations of Current Techniques.....	2-1
Classical Approximation Techniques.....	2-1
Collocation via Goal Programming.....	2-2
The Genetic Algorithm.....	2-5
III. Methodology	
Comparison of Compromise Programming Models.....	3-1
Generalized Compromise Programming.....	3-1
Modified Compromise Programming.....	3-6
Solution Process.....	3-8
Genetic Algorithm Parameter Settings.....	3-9

Measures of Effectiveness .....	3-10
Quantitative Measures .....	3-10
Qualitative Measures .....	3-10
 IV. Worked Examples	
Navier - Stokes Equation .....	4-1
Problem Description .....	4-1
Solution and Analysis .....	4-2
Linear Boundary Value Problem .....	4-4
Problem Description .....	4-4
Solution and Analysis .....	4-6
The Steady Burgers Equation .....	4-7
Problem Description .....	4-7
Solution and Analysis .....	4-9
The Steady Fisher Equation .....	4-13
Problem Description .....	4-13
Solution and Analysis .....	4-14
 V. Observations and Recommendations	
Observations .....	5-1
Validation of Ng's Work .....	5-1
Modified Compromise Programming .....	5-1
Genetic Algorithms .....	5-1
Weighted Boundary Residuals .....	5-3
Collocation Theory .....	5-3
Choice of Collocation Points .....	5-4
Choice of Test Functions .....	5-5
Recommendations for Further Study .....	5-5
Choice of Collocation Points .....	5-5
Error Analysis .....	5-6
Termination Criteria .....	5-6
Other Classes of Problems .....	5-7
Conclusion .....	5-7
 Appendix A: Genetic Algorithms .....	A-1



Appendix B: Sample Computer Code.....	B-1
References.....	REF-1
Vita.....	.Vita-1

## List of Figures

Figure	Page
3.1 Epsilon Ball.....	3-6
4.1 Velocity distribution of flow at a stagnation point.....	4-2
4.2 Velocity distribution of Navier-Stokes solution using ten evenly spaced collocation point model.....	4-4
4.3 Comparison of exact solution of the linear boundary value problem with solution from the 100 point model.....	4-7
4.4 Graph of exact solution to the Burgers equation for two values of $\nu$ .....	4-9
4.5 Comparison of the Burger equation exact solution with approximate solution from 100 point model.....	4-11
4.6 Graphical comparison of the Burgers exact solution to the solutions obtained via the evenly and unevenly spaced ten point models.....	4-12
4.7 Graphical comparison of solutions to the Fisher equation via (a) 10 pt. (even) ; (b) 20 pt. ; (c) 100 pt.; and 10 pt. (uneven) models.....	4-16
4.8 Graph of energy function for the 20 point model (Fisher equation).....	4-17

## List of Tables

Table	Page
4.1 Quantitative results for the solutions to the Navier Stokes equation.....	4-3
4.2 Coefficients of the trial functions for the Navier Stokes solutions.....	4-4
4.3 Quantitative results for the solutions to the linear boundary value problem.....	4-6
4.4 Coefficients of the trial functions for the linear boundary value problem.....	4-7
4.5 Quantitative results for the solutions to the Burgers equation.....	4-10
4.6 Coefficients of the trial functions for the steady Burgers equation.....	4-12
4.7 Quantitative results for the solutions to the Fisher equation.....	4-15
4.8 Coefficients of the trial functions for the steady Fisher equation.....	4-17

## **Abstract**

This study is an extension of Ng's previous work in which goal programming was used to determine an approximate solution to a boundary value problem. This approach follows the same basic approach developed by Ng in which the method of collocation was recast as a compromise programming model. Hence, instead of solving a system of simultaneous nonlinear equations, one seeks a compromise solution which minimizes (in a weighted residual sense) a vector norm of the differential equation residuals. A difference in this approach is that it makes use of a genetic algorithm as the optimizing engine as opposed to the pattern search used by Ng. The model developed in this approach also consists of a modification to the generalized compromise programming model by eliminating the need for deviation variables. As a result, this model is a simplification in that the number of decision variables is completely independent of the number of collocation points. This technique is very robust in that it has been shown to produce good results on a variety of problems. The results of four example problems compare favorably with those of Ng and other solution techniques.

# A ROBUST METHOD OF SOLVING NONLINEAR BOUNDARY VALUE PROBLEMS VIA MODIFIED COMPROMISE PROGRAMMING

## I. Introduction

### Background

Projection methods encompass a whole class of techniques to approximate the solution of boundary value problems. A particular projection method that is widely used is the method of collocation. In this method, the solution is approximated by a test function which is a finite sum of terms which are referred to as trial functions. This test function is generally constructed to be linear with respect to its unknown coefficients or parameters and is usually of the form

$$v = \sum_{i=1}^N c_i \cdot \phi_i$$

where  $\{c_i\}$  is the set of unknown parameters and  $\{\phi_i\}$  is the chosen set of trial functions.

The test function  $v$  is considered a solution of the boundary value problem of the form

$$F(x, u, \frac{d^2u}{dx^2}, \dots, \frac{d^nu}{dx^n}) = 0 \quad (1-1)$$

$$Bu = 0 \quad \text{on boundary,} \quad (1-2)$$

where  $F$  is a function of its arguments and  $B$  is a boundary operator, provided  $v$  satisfies Equations 1-1 and 1-2 for some choice of  $\{c_i\}$ . Substituting the test function into Equation 1-1 yields the following set of residual equations.

$$R(c, x) = F(x, \sum_{i=1}^N c_i \cdot \phi_i, \sum_{i=1}^N c_i \cdot \frac{d\phi_i}{dx}, \dots, \sum_{i=1}^N c_i \cdot \frac{d^n \phi_i}{dx^n}) = 0 \quad (1-3)$$

Choosing  $N$  distinct collocation points,  $\{x_i\}$ , and requiring that the residual equation be exactly satisfied at each collocation point yields a system of equations (one equation for each collocation point). An important aspect of this technique is that once the collocation points,  $\{x_i\}$ , are chosen, they are no longer variables with respect to the system of equations to be solved. Instead, the coefficients,  $\{c_i\}$ , are the only unknowns in the system. Solving the equations for these unknown coefficients and substituting those values into the original trial function will yield an approximation of the solution to the differential equation. This technique yields a solution that interpolates the differential equation at the collocations points only. In general, this approximation will not interpolate the differential equation at other points in the domain nor will it satisfy the boundary conditions. To ensure the boundary conditions are satisfied, residuals similar to Equation 1-3 must be formed. To prevent the system of equations from being over specified, either the number of collocation points or the number of trial functions must be adjusted accordingly. The point here is that in standard collocation methods the number of equations should equal the number of unknown coefficients  $\{c_i\}$ .

For linear differential equations, the set of residual equations reduces to a system of linear equations which must be solved. Consequently, the method is very simple to use and requires relatively minimal computing resources. However, in the case of a nonlinear differential equation, a drawback to the method is the requirement to solve a system of

nonlinear equations. Normal procedures might use Newton's method or another similar technique to solve for the  $\{c_i\}$ . For relatively small systems of equations (i.e., only a few collocation points) this may be a relatively easy problem to solve. However, if a more accurate solution is desired more collocation points must be used thereby increasing the dimension of the system. While large systems of linear equations can be relatively easy to solve, the same cannot be said of nonlinear equations. Hence, a technique which simplifies the solution process for the system of equations would be desirable.

Kevin Ng proposed a modification to the collocation technique by utilizing a multi-criteria decision making approach to solve for the unknown coefficients (17:103). Rather than solve for the coefficients by requiring the residuals equal zero at the  $N$  collocation points, Ng developed a goal programming model where the objective was to minimize the sum of the absolute value of the residuals,  $|R(c_i, x_j)|$ , over  $M$  collocation points. This approach allows one to choose  $M$  much greater than  $N$  without having to add more trial functions (and thus more  $c_i$ 's) to the test function. Thus we may end up with a more "compact" approximation of the unknown function.

The general form of his model is as follows:

$$\min_{c_i, i=1,2,\dots,N} \sum_{j=1}^M |R(c_i, x_j)|. \quad (1-4)$$

subject to

$$R(c_i, x_1) + n_1 - p_1 = 0, \quad (1-5)$$

$$R(c_i, x_2) + n_2 - p_2 = 0,$$

$$\vdots$$

$$R(c_i, x_M) + n_M - p_M = 0,$$

where  $\{n_j\}$  is the set of deviational variables introduced to represent negative deviations from goal level  $j$  and  $\{p_j\}$  is the set of deviational variables representing positive deviations from goal level  $j$ . It follows that each of these deviational variables must be non-negative. In the goal programming construct of the model the objective function will actually take on the form:

$$\min \sum_{j=1}^M (n_j + p_j)$$

Upon solving this optimization problem, Ng determined the root mean square of the deviations over selected points in the domain. If that value is greater than a specified tolerance level, an additional trial function is added to the test function and the model is run again. The use of the root mean square is based on Ng's observation that in the absence of rigorous error bounds on the approximate solution, users of collocation methods assume that a small root mean square is an indication of a good solution (17:106). This process continues until the root mean square is within tolerance. Typically, Ng used a tolerance value of 0.1 for the root mean square as his standard.

To handle the nonlinearity of the constraints in the above model, Ng used a modified pattern search algorithm based on the method developed by Hooke and Jeeves (17:106). This algorithm produced results on several test problems that were very good compared to solutions achieved by more classical solution procedures.

## **Problem**

While Ng developed a useful new approach to solve boundary value problems, there is room to improve on his technique. As shall be seen, his method can be improved in two ways. First, some improvement can be made in reducing the problem size. By this I am referring to the dimensionality and, hence, the number of variables utilized to generate a



solution. The nature of his goal programming formulation calls for adding two deviational variables for each collocation point over which he minimized. A method which introduces no new variables would be a clear improvement over his technique. A second improvement can be made in the robustness of his approach. Here, I am referring to the optimization algorithm used to find the optimal solution. While the Hooke and Jeeves pattern search has been shown to be useful in many nonlinear programming applications, it has some of the same limitations as other optimizing algorithms. The primary limitation of the algorithm is the uncertainty involved in making the proper selection of a starting point. This matter will be discussed in more detail in Chapter 2.

I will present a simpler yet more robust method of approximating the solution of a steady partial differential equation via collocation and modified compromise programming. Here I have used the more general framework of compromise programming (which is an extension of goal programming) as it will be a more appropriate label for the model.

## **Scope**

While the possibilities of implementing the solution technique are many, this research is limited to the treatment of scalar partial differential equations. Within this class of problems I shall demonstrate the applicability of the procedure to linear and nonlinear problems.

This thesis presents a graduated approach to the solution of four different boundary value problems. First, I shall demonstrate the technique on the Navier-Stokes equation which was also solved by Ng. This problem will allow for a direct comparison of the two techniques. Next, I will solve a simple linear ordinary differential equation. This equation will be solved to demonstrate the applicability of the technique to other classes of problems and is not a problem of any physical significance. I will then present the

solutions to two classic nonlinear problems: the steady Burgers equation and the steady Fisher equation. In each problem I will compare my results with analytic solutions (when available) and other approximating solutions via various other techniques in the field.

## **General Approach**

As previously stated, the goal of this research was to work with Ng's basic problem formulation and modify it in order to simplify the model while incorporating a more sophisticated optimization algorithm. I have been able accomplish both of these objectives by relying on a genetic algorithm as the optimizing engine. As it turns out, the choice of a genetic algorithm almost necessitates the simpler model construct. Because genetic algorithms cannot handle explicit constraints, Ng's constrained model had to be formulated into a model with no constraints. While such techniques as the introduction of penalty functions or barrier functions transform a constrained problem into an unconstrained one, I found that this was not necessary. A simpler transformation can be performed that provides the same effect without undue complication of the model. Further, this transformation eliminates the need for the deviational variables that were introduced into the goal programming model. As a result, the model is of a more compact and intuitive form. For example, Ng's approach with a four term test function and one hundred collocation points would require a model with over two hundred variables and at least one hundred constraints. My formulation of the same problem has only a single objective function of four variables and no constraints. The advantage here is obvious as long this model produces results that are at least as good as Ng's.

## **Presentation**

This report is organized as follows. Up to this point, the background for the problem has been laid as well as the general direction for the solution. Chapter Two is devoted to presenting preliminary information that will serve to justify the need for the research as well as validate my approach. It will also serve as a point of departure from current knowledge in the field to my research. Chapter Three will present the methodology of my research. This chapter will include a detailed description of compromise programming and will serve to explain the difference between generalized and modified compromise programming. Chapter Four will consist of worked examples and analysis of the technique to include a detailed description of each problem solved as well as any known solutions to the problems. Chapter Five will conclude with general findings and recommendations for further study.

## **II. Preliminaries**

### **Introduction**

The purpose of this chapter is two-fold. First, it will present various approaches to solving boundary value problems (including the method developed by Ng) all of which have some limitations. Second, further review of literature in the field will demonstrate where my approach may be able to overcome some of those shortcomings. As a result, my technique will be presented as an alternative method to be utilized if the limitations of the other techniques prove to be an obstacle for a particular problem or set of problems.

### **Limitations of Current Techniques**

**Classical Approximation Techniques.** In general, when analytical approaches to solving differential equations are inadequate, some sort of approximation technique is utilized. Significant texts by Prenter (20); Villadsen and Michelsen (29); Ascher and Russell (1); Powell (20); and numerous others are devoted exclusively to approximation theory and applications. The method of weighted residuals (MWR) is one such approximation technique that stands at the forefront (17:103). This technique may use any one of a number of means to "explain" the inherent error in the approximation process. Collocation, orthogonal collocation, least squares, the abdominal method, and the Galerkin method are some of the primary means of establishing the error criteria (17:103). Although these techniques are often useful, they also have drawbacks that may leave the engineer in want of another approach.

All but the collocation methods share the common need to evaluate integrals of the chosen trial functions (17:103). The need to evaluate these integrals is the main

drawback of such techniques. If the trial function utilized is not easily integrable, the solution process is hindered. While there are numerical techniques available to approximate the integration process, automation of such techniques adds an additional burden on the solver. In addition, using approximation techniques within other approximation techniques may hamper the goal of meeting specified error criteria.

Although the collocation technique avoids the problems involved in integrating the trial functions, it has another problem when being utilized to solve a nonlinear differential equation -- the problem of solving systems of nonlinear algebraic equations. While techniques abound in solving nonlinear problems such as Newton's method, the bisection method, the steepest ascent method, etc., each has its own limitations. Often the use of these methods is exhausting in terms of resources spent. One reason for this shortcoming is that the success or failure of the technique is often predicated on the choice of the starting point for the search. This choice is not a trivial matter and may ultimately determine the success or failure of the search. In addition, such search techniques may be limited by restrictive assumptions concerning the problem domain and nature of the underlying function. When assumptions pertaining to continuity, existence of derivatives, and unimodality of the functions cannot be reasonably applied, some of these techniques can no longer be utilized. Lastly, since these methods may involve the inversion of large matrices other problems may arise. Such things as the ill-conditioning of the matrix, definiteness, and singularity all can cause problems that would require special attention (15:161-5). In addition, the computing and storage requirements for such operations can impose a burden on the hardware utilized. Thus, no single method may be completely applicable to every class of problem.

**Collocation via Goal Programming.** In his first paper on the subject, Ng applied a technique common to the operations research community to address some of

these shortcomings (17). While relying on the basic collocation method as the approach to approximating the solution to the differential equation, he utilizes goal programming as the means to determine the unknown coefficients of the trial functions. In doing so, he formulates a mathematical model that minimizes (in a weighted residual sense) the sum of the absolute value deviations of the differential equation residual (17:104). Thus he has taken the general procedure of searching for roots of simultaneous nonlinear equations and transformed it into a procedure of optimizing a nonlinear objective function with associated constraints in the form of goals. The overall objective then is to minimize the sum of absolute value deviations over the chosen collocation points.

Ng's novel approach has several advantages. First, his technique is very general in that it allows him to choose more collocation points than he has terms in his set of trial functions. The advantage here is that he can approximate the differential equation across a larger subset of the domain without having to concomitantly add a like number of terms to his approximating function (17:104). Another advantage is that his technique does not require integration. Lastly, by utilizing a directed pattern search algorithm in his goal programming model, he avoids the programming effort and other pitfalls inherent in using a Newton search or other like methods (17:104).

While Ng's use of a pattern search as the optimizing vehicle for the model does avoid certain problems, it does have limitations. He used an approach developed by Hooke and Jeeves in 1960 (10) with slight modifications implemented by Ignizio (12). The nature of this algorithm is to explore trial solutions in specified step sizes from an initial base point and sequentially seek "better" solutions in incremental step sizes. These step sizes can be increased or decreased based on the criteria established by the programmer. The search is halted when a given number of steps does not yield marked improvement as defined by the programmer. While this algorithm adequately handles the nonlinear properties of Ng's example problems, certain aspects of its methodology may limit its usefulness in

other applications. For example, since the pattern search (like other methods previously mentioned) begins its search from a single base point (10:215), it suffers the same drawbacks as those other techniques when the starting point is not suitably chosen. Convergence to an optimum cannot be guaranteed and, when achieved, there is no guarantee of global optimality. Ng handles this problem by sequentially adding more terms to his approximating function when the solution obtained by the pattern search is not within a specified tolerance. One could reasonably expect that this method might generate an unnecessarily large number of terms in the approximating function if the pattern search never converges to a point near the global optimum. Thus, while Ng's approach has shown success there may be occasions when a more robust approach is desired.

At this point one might justifiably ask: Is there one best algorithm to use for nonlinear programming problems? Obviously, this is a loaded question and the answer depends not only on what one means by "best" (i.e., rate of convergence, computational complexity, etc.) but also on the nature or structure of the problem. I have chosen not to make an issue of such things as rate of convergence or computational complexity for a few reasons. First, often the differences among the different techniques is minimal and given the power of modern computing resources a "slow" algorithm is not necessarily a bad algorithm as long as convergence is achieved in a reasonable amount of time. Also, with the advent of viable non-deterministic algorithms (such as genetic algorithms, simulated annealing, and other evolutionary based techniques), making such comparisons is difficult if not impossible. Thus, it seems reasonable to concentrate on a solution technique that is not severely limited by the particular structure type.

There exists a great amount of literature on various methods of nonlinear programming. Since most of the methods are covered in standard text books (which also happen to give the best surveys of the subject), I have focused my research as such.

Perhaps Ignizio provides the best summary of nonlinear programming techniques. He reported the following well established observations:

1. A particular method may perform well on one problem but poorly on a slight modification to that problem.
2. The results obtained by any method are highly dependent on the starting point or points used to initiate the search.
3. One can only hope to obtain a local optimal unless the problem is of very special form
4. The vast majority of the rather small amount of computational experience available has been addressed toward problems of a very small, if not trivial, size.

One should not naively accept the conclusions, as drawn by some investigators, with regards to their choice of algorithms as based on such minimal experience as cited above (12:155). It should be noted that his observations are somewhat dated (1976) and, hence, do not take into account newer algorithms. It shall be seen that the use of a genetic algorithm may avoid some of the problems mentioned above.

## **The Genetic Algorithm**

Now we turn our attention to the use of a genetic algorithm to build on the goal programming foundation laid by Ng. Since Ng has already established the success of a pattern search algorithm in the model, it is important to show how a genetic algorithm can improve the solution process by overcoming some of the shortcomings cited by Ignizio. The explanation of how genetic algorithms work is better left to an appendix and will be treated as such (see Appendix A). For now it should suffice to explain the applicability of the genetic algorithm to the problem at hand.

The genetic algorithm (GA) is a fairly recent development that has drawn increasing interest over the past few years (27:17). First introduced by John Holland in 1975 (11) ,



GAs are now widely recognized as very robust mechanisms in optimization applications (7:10). These algorithms, which owe their name to the analogous study of evolution, have been very useful in such areas as biological modeling (5:872), least squares curve fitting (14:8), gas pipeline control systems (7:125), and spin glass models (28:549), to name but a few. This very wide range of current GA usage suggests that they might also have applicability in the area of approximating solutions to differential equations.

One word that has often been used to describe genetic algorithms is robust. This adjective describes not only their versatility in a wide range of applications but also the manner in which they adapt within a particular application (7:1). For example, in a complex optimization problem with many stationary points, GAs are not usually “fooled” by local optima and, when implemented correctly, stand a high probability of locating the region of global optimality (7:74). Convergence to such local optimal points was one of the drawbacks mentioned in the previous section when referring to other techniques.

Goldberg provides much insight into the utility of GAs. In his book (7), Goldberg provides an excellent treatment of how GAs differ from other conventional optimization procedures. These differences also serve to demonstrate the versatility of GAs and how they overcome some of the traditional shortcomings cited by Ignizio. Some of the differences he mentions are

1. GAs code the parameter set as opposed to the parameters themselves.
2. GAs search from a population of points rather than a single point. This allows for more breadth in the search. Many algorithms will never converge to a global optimum unless given the proper starting point.
3. GAs use objective function information directly rather than derivatives, gradients, directions of maximum ascent, etc. (7:7).
4. GAs use probabilistic transition rules as opposed to deterministic ones. This feature is to be contrasted with those “hill-climbing” techniques that will

always follow the direction of steepest ascent, regardless of where the global optimum lies (7:7).

These differences are central to the robustness of the algorithm. For example, by coding the parameter set, GAs make use of the coding similarities that exist among those parameter values that yield the most optimal result. The GA then builds on this information and continues to exploit further similarities among the most optimal parameter values. Because GAs work from a population of search points that can number in the hundreds, they do not suffer the setbacks of those techniques that search from a single point. Lastly, because GAs do not depend on auxiliary information about the objective function, they can be applied to the most pathological cases where the requirements of continuity and the existence of derivatives cannot be assured.

One of the major findings reported by Goldberg concerns the genetic algorithm's success in problems where numerous local optima exist. He found that while GAs will not necessarily guarantee convergence (like some other algorithms), they do, however, sort out areas of space quickly. In doing so they are able to ferret out those regions of best parameters that are sometimes never even looked at by more traditional methods. Goldberg implies that by finding these regions, the GA will at least settle on an area that is better than those found by other methods (7:74). Convergence to the absolute optimum, though desirable, may not be necessary provided the GA reaches an area that is more optimal than the best local optimum found by other search methods. Reaching such an area could very well prevent Ng's method from adding unnecessary terms to the approximating function as previously described. Thus, one might expect that GAs have the potential to streamline the collocation solution technique of Ng.

The genetic algorithm is not without its drawbacks. One drawback is that the algorithm, like some others, does not handle constraints explicitly. Rather, a constrained

problem may need to be modeled with penalty functions to ensure feasibility. At least two genetic algorithm software packages have been developed that automatically generate these penalty functions. Another weakness of the genetic algorithm is that it is not a particularly fast technique. Many genetic operations must be performed on each generation. Additionally, the algorithm must constantly code and decode the parameters as well as perform the mathematical operations inherent to the optimization problem on each individual in the population. Hence one can see that the algorithm may require more computational time than other procedures. Consequently, much work is being done to implement genetic algorithms on parallel processors. Additionally, although genetic algorithms do sort out areas of space very efficiently, they offer no guarantee of convergence. There is a class of problems that are considered genetic algorithm deceptive. Such problems result when the optimum solution is "surrounded" by the most suboptimal solutions. In this event the algorithm is likely to converge prematurely to a suboptimal point (7:45). Nevertheless, the countless applications where genetic algorithms have been used successfully suggest that these limitations are not insurmountable.

The last point to be highlighted in reference to GAs is that they have already been successfully utilized in constrained mathematical programming models. One approach that has been successfully utilized has been to code the constraints as penalty functions within the objective function (21:191). A properly chosen penalty function will act exactly like a constraint by imposing a large penalty on the objective function when the constraint is violated. As an extension to single objective function optimization, Schaffer successfully solved problems with multiple objectives (23:99). Since multiple objective function models are essentially the same as goal programming models, Schaffer's work provides strong evidence that GAs can be successfully implemented in my research. As it turns out, my model formulation uses a variant of the penalty function method but in a

much simpler format. Hence it seems reasonable to conclude that the genetic algorithm is a viable alternate optimizing engine for the model.

### III. Methodology

#### Comparison of Compromise Programming Models

**Generalized Compromise Programming.** As previously mentioned, compromise programming is an extension of goal programming -- both of which fall under the general heading of multiple criteria decision making. In the typical multiple criteria decision making problem, a decision maker has a multitude of objective functions each competing for the same resources. Ideally, the decision maker would like to maximize ( or minimize, as the case may be) the outcome of each of his objectives. Seo's representation of the mathematical formulation of this problem is as follows:

$$\underset{x \in X}{\text{maximize}} \quad f(x) \equiv (f_1(x), f_2(x), \dots, f_m(x))$$

where  $f_1(x), \dots, f_m(x)$  are  $M$  different objective functions and  $X$  is the feasible region (26:59-60). Since numerous objectives may compete for the same resources, some trade-off must ensue. The goal programming strategy to solve such a problem is to put it into a goal setting context. In this formulation, the decision maker sets goals for each of his objectives and seeks a satisficing solution that gets him as close to each of his goals as possible. Thus, a distance metric must be introduced to represent the closeness of an objective to its goal. The mathematical representation of this model is as follows:

$$\underset{x \in X}{\text{minimize}} \quad d(f(x), \hat{f})$$

where  $\hat{f} = (\hat{f}_1, \dots, \hat{f}_M)$  is the vector of aspiration levels for the  $M$  goals and  $d(\cdot, \cdot)$  is the non-negative distance from the vector  $f(x)$  and its goal vector,  $\hat{f}$  (26:60). The natural way to measure the distance between the two vectors is via a norm.

In the compromise programming approach to the same problem, the goal vector,  $\hat{f}$ , is simply replaced by an ideal vector,  $f^*$ , which represents the maximum possible value for each  $f_j(x)$ . Thus, each aspiration level is actually the best possible outcome for the corresponding objective. Hence, any solution (known as a compromise solution) is measured in terms of its closeness to the ideal (30:314).

In order to facilitate the mathematical modeling of the above problems, they each must be put into an acceptable format. To facilitate this, the set of deviational variables  $\{n_j, p_j\}$  is introduced. In the goal programming construct these variables are defined as follows:

$n_j \equiv$  amount by which goal  $j$  is not met  
 $p_j \equiv$  amount by which goal  $j$  is surpassed

In other words, they represent the under-achievement or over-achievement of the  $j$  goals. Hence for a given goal,  $j$ , if both deviational variables are zero, then the goal is exactly satisfied. Each goal of the form

$$f_j(x) = 0 \quad (3-1)$$

is then represented as a separate constraint in the model as follows:

$$f_j(x) + n_j - p_j = 0 \quad (3-2)$$

Hence, each goal is converted from an objective function to a constraint of the form in Equation 3-2. It follows that the dimension of the problem increases as goals are added since each goal represents two additional variables in the model.

In the generalized linear programming format of this model, all decision variables must assume non-negative values, thus necessitating a separate variable for positive and negative deviations. However, if an algorithm without non-negativity requirements is utilized, then only one deviation variable,  $d_j$ , would be needed for each goal level  $j$ . It follows that  $d_j$  has no sign restriction and can, therefore, represent either under-achievement or over-achievement of the goals depending on the sign that it assumes.

At this point it is important to understand two basic schools of thought in reference to goal programming. The format which has just been presented is sometimes known as non-dominated goal programming. It carries this label because the technique will render a non-dominated or Pareto optimal solution. Within this technique, the relative importance of a particular goal in comparison to another goal may be represented by a weighting scheme. In such a scheme the goals with more importance are assigned higher weights. The other technique is lexicographic or preemptive goal programming. In this framework, the decision maker assigns priority levels to each of the goals. In such a strategy some goals may be designated as absolute -- meaning that they must be satisfied. In the general mathematical programming framework, an absolute goal is simply a normal constraint that must be satisfied for the solution to be feasible. Hence a solution where the  $p_j$  value for an absolute goal is non-zero would mean that the absolute goal is unattainable. In some cases there may be no need for absolute goals. Goals of lesser priority will be satisfied if possible but not at the expense of sacrificing a goal with a higher priority. If a goal with lower priority cannot be satisfied, then the solution procedure should attempt to come as close to satisfying that goal as possible. Here

closeness takes on the same sense as described above and is, hence, measured via a vector norm.

In Ng's initial published research in this area (17) it is unclear as to which technique he utilized. In a subsequent paper (18), he clearly used the preemptive goal programming approach. As will become evident, I chose to utilize the non-dominated approach for this research.

Regardless of the goal programming technique implemented, the objective function will incorporate some form of a distance metric. Before this aspect is discussed in detail, it is important to consider which metric is most suitable to use. The definition of a discrete  $l_p$  norm to measure the distance  $d_p(x, y)$  is given by the following:

$$d_p(x, y) \equiv \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} \quad (3-3)$$

where  $x$  and  $y$  are  $n$ -dimensional vectors and  $1 \leq p \leq \infty$ . Typically, either the  $l_1$ ,  $l_2$ , or  $l_\infty$  norms are utilized. While in the finite-dimensional case it can be shown that these three norms are equivalent, there may be good reason to use one over another. For example, in least squares curve fitting (which is a compromise program of special structure) the  $l_2$  norm is utilized for computational convenience if one can assume that errors are normally distributed random variables (2:52). The  $l_\infty$  norm (also known as the max norm) which is defined as follows

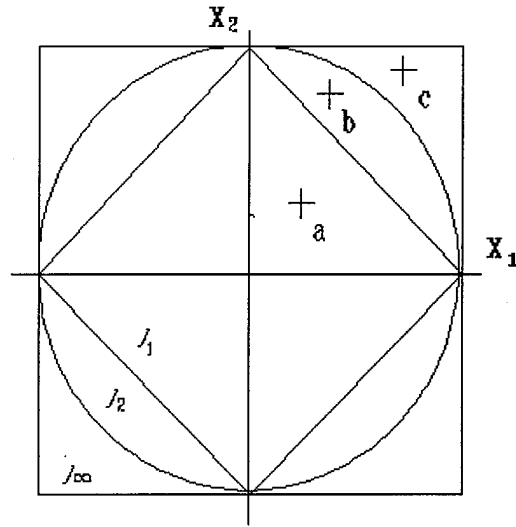
$$d_\infty(x, y) \equiv \max_{1 \leq i \leq n} |x_i - y_i| \quad (3-4)$$

is used when the decision maker exhibits a totally non-compensatory preference structure. In other words, he is only concerned with minimizing the maximum distance from any one goal.



These examples raise the question: "Is there a best  $l_p$  norm to use for the problem at hand?" This research concludes that, of the three, the  $l_1$  norm is the most suitable choice for this problem. To justify this choice, consider the 2-dimensional "balls" of radius  $\epsilon$  centered at the origin with respect to  $l_1$ ,  $l_2$ , and  $l_\infty$  norms, as shown in Figure 1. The boundaries of a ball in this figure correspond to points where the respective norms are equal to  $\epsilon$ . For example, the points on the curve for  $l_1$ , represent all of the possible vectors  $(x_1, x_2)$  where  $d_1((x_1, x_2), (0, 0))$  equals  $\epsilon$ . Likewise, this is also true for the  $l_2$  and  $l_\infty$  norm curves. Let us assume that we are trying to minimize the distance  $d_p((x_1, x_2), (0, 0))$  and the termination criteria for our algorithm is that this distance be less than  $\epsilon$ . If the  $l_\infty$  norm is chosen as the metric, it is obvious that although each of the points marked "a", "b", and "c" all meet this criteria, point "a" clearly represents the vector that is closest to the vector  $(0, 0)$ . It is the non-dominated choice of the three. However, if the  $l_1$  metric is utilized, only point "a" represents a vector that meets the  $\epsilon$  tolerance that was established. Thus in the finite-dimensional case for  $1 \leq p \leq \infty$ , the  $l_1$  norm is the most restrictive and represents the metric that is more likely to yield a truly non-dominated solution. This conclusion is generally well accepted and was shown in several empirical studies done by Barrodale where he was conducting curve fitting experiments (2:56). He found that in every case the  $l_1$  norm provided the best fit for his data sets.

It is important to note that this conclusion is equally applicable for n-dimensional space since the two-dimensional ball can be generalized to n-space. It is also interesting to note, if one is trying to minimize the distance  $d_p(f(x), g(x))$ , where  $f$  and  $g$  represent continuous functions on a compact set, just the opposite is true. In this instance it can be shown that the continuous  $L_\infty$  norm is most suitable. Although this study does concern the approximation of functions which are likely to be continuous, the solution technique involves a discrete-valued function at a finite number of points. Hence the discrete norm is appropriate.



**Figure 3.1 Epsilon Ball**

The last aspect of the model which requires discussion is the form of the objective function. Since the objective of a goal programming model is to minimize the amount by which the goals are not achieved, the specific form of the objective function is as follows:

$$\underset{x \in X}{\text{minimize}} \quad \|n + p\|_p \quad (3-5)$$

where  $n$  and  $p$  are the vectors of deviational variables representing the amount by which the goals are missed. For  $p = 1$ , this representation is equivalent to minimizing the  $l_1$  distance metric of these vectors relative to the zero vector.

**Modified Compromise Programming.** As described in Chapter One, Ng's model was based on minimizing the residual equations at each of the collocation points. Thus, each residual equation was a goal. Naturally, the goal was for each residual to be

exactly zero. I have chosen this multi-objective model to be labeled a compromise programming model (as opposed to a goal programming model) since having a residual equate to zero represents an ideal that cannot be bettered. However, this model differs from Ng's in a significant way other than its name.

Recalling that the use of a genetic algorithm necessitates transforming the constrained model into an unconstrained model, I had to seek a way to do so. It became apparent that by eliminating the deviation variables from the residual equations and simply minimizing the norm of the vector of residual equations would represent a completely equivalent formulation of the same problem. Hence, the objective function of this modified compromise programming model is as follows:

$$\underset{c_i, i=1,2,\dots,N}{\text{minimize}} \sum_{j=1}^M |R(c_i, x_j)|. \quad (3-6)$$

Although this is an equivalent representation of the generalized model, it is modified in the sense that both the deviational variables and constraints have been eliminated.

In this study, treatment of the boundary conditions must also be considered. It turns out that this involves only a slight modification to the model. While Ng was able (in some instances) to choose test functions that would automatically satisfy homogeneous boundary conditions regardless of the number of terms, this may not always be easy. In such a case, all that needs to be done is to form additional residual equations corresponding to the boundary conditions. For example, a boundary condition such as the following

$$u'(0) = 0 \quad (3-7)$$

would be transformed into the following residual equation

$$RB(0) = \sum_{i=1}^N c_i \frac{d\phi_i}{dx}(0) = 0 \quad (3-8)$$

This boundary residual would then be added to the other residuals to form the complete objective function. For rigid boundary conditions, Ng's lexicographic approach would necessitate considering the boundary conditions as absolute goals requiring additional constraints. In the modified compromise programming approach, the rigidity can be simulated by incorporating a positive valued multiplicative weighting factor into the boundary residual, Equation 3-8.

The choice of the  $l_1$  norm is easily incorporated into the genetic algorithm. This is a somewhat subtle advantage of the algorithm since I am minimizing over a non-differentiable function (due to the absolute value). Special modifications to other "hill climbing" algorithms would be required because of this fact.

## **Solution Procedure**

For each of the problems that have been solved, the following iterative procedure was used.

- 1.) Choose ten evenly spaced collocation points over which to minimize and an initial three term test function. (i.e., set  $i = 3$ )
- 2.) Execute modified compromise program model minimizing over the  $l_1$  norm of the residual vector. If the residual norm at iteration  $i$  is at least ten percent smaller than the residual norm at iteration  $i - 1$ , then proceed to step three. Otherwise, stop.
- 3.) Add another trial function and return to step two.

This procedure was also used for a twenty point and a one hundred point collocation model. Additionally, it was executed with a ten point collocation model where the spacing of the points was adjusted so as to emphasize key areas of the domain such as around boundaries and in the vicinity of a shock.

## **Genetic Algorithm Parameter Settings**

Numerous studies have been done to determine if there is such a thing as optimal parameter settings for genetic algorithms. Most research has considered varying the population size, cross-over rate, and mutation rate (see Appendix A). This work has produced no universally accepted specific results. In fact some of the research directly contradicts other research while some other studies have produced unrealizable results. For example, one particular analytical approach concluded that the optimal population size for a given problem was zero (6:4). It seems apparent that the stochastic nature of the algorithm makes such analysis difficult if not impossible. When the studies have produced specific results, these results tend to apply only to a certain problem and cannot be universally applied to all problems. However, some useful conclusions can be made on the ranges of the parameter settings.

A detailed study performed by Schaffer indicated that good performance can be achieved with population sizes of twenty to thirty, cross-over rates ranging from 0.75 to 0.95, and mutation rates from 0.005 to 0.01 (24:55). Another study he performed indicated that the use of Gray coding improved performance by eliminating Hamming cliffs (see Appendix A for a description of Gray coding and Hamming cliffs) (24:59).

In consideration of these studies I utilized Gray coding and ran each model with three distinct sets of parameter settings as follows

Setting 1: Population size = 20; Cross-over rate = 0.75; Mutation rate = 0.005 .

Setting 2: Population size = 30; Cross-over rate = 0.95; Mutation rate = 0.01

Setting 3: Population size = 20; Cross-over rate = 0.90; Mutation rate = 0.075

The software I utilized, Genesis (9), also permitted the use of an elitist strategy wherein the most fit individual of each population is automatically selected to reproduce. This computer program was implemented on a *Sun Sparcstation 20* computer.

## Measures of Effectiveness

**Quantitative Measures.** In determining the validity of the results for each problem studied several quantitative measures have been utilized. The first is simply the  $l_1$  norm of the residual vector over ten evenly spaced points. If this norm is on the order of  $10^{-1}$ , then this would indicate a good solution. When an exact solution for a problem is available, I shall also present the  $l_1$  norm of an error vector measured over ten evenly spaced points. Here, the term "error" indicates the absolute difference between the exact solution,  $f(x)$ , and the approximated solution,  $v(x)$ , over those ten points. This metric (as opposed to the residual metric) actually gives a better indication of the quality of the solution since the overall goal is to get an approximation of the exact solution.

Additionally, I compared critical values of my solution to other known exact or approximated solutions. For example, in the Navier - Stokes problem, an empirical solution determined by Froessling indicated that the value of the wall sheer stress  $f''(0)$  equals 1.312 (25:98). Other such comparisons may be available on the various problems.

**Qualitative Measures.** The primary qualitative measure of effectiveness will involve analyzing the graph of the solution. Ideally, this graph should compare well with the graph of an exact or other well accepted approximated solution.

## IV. Worked Examples

### Navier - Stokes Equation

**Problem Description.** This problem was the first one presented by Ng in his original paper. The Navier - Stokes equations model fluid flows and generally include equations that govern mass conservation and momentum balance. For the case of 3-dimensional, axi-symmetric fluid flow with stagnation, where the fluid strikes a surface perpendicularly and is allowed to flow away in all directions, the Navier - Stokes equations can be greatly simplified resulting in the following dimension-less differential equation (25:101):

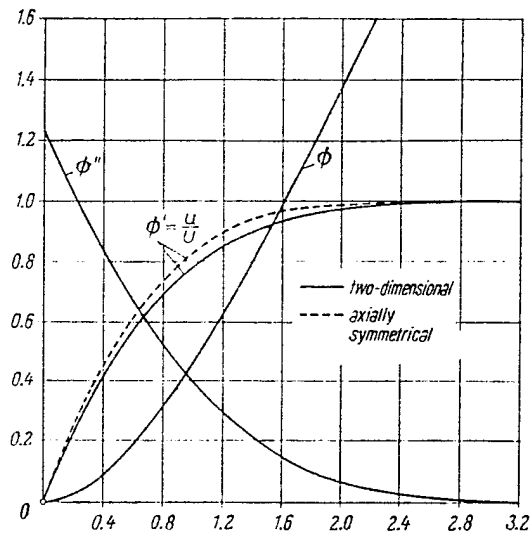
$$f''' + 2 f f'' + 1 - f'^2 = 0 \quad (4-1)$$

with boundary conditions as follows:

$$\begin{aligned} f(0) &= f'(0) = 0 \quad \text{and} \\ \lim_{x \rightarrow \infty} f'(x) &= 1. \end{aligned} \quad (4-2)$$

on the interval  $(0, \infty)$ .

Although no solution to this problem was available, a graph of the function as well as tabular data taken from a paper by Froessling (25:98-100) will allow for direct comparison of the solutions. For example, Ng cited Froessling's empirically determined value of  $f''(0) = 1.312$  as a validation point. Additionally, the dashed line in Figure 4.1 represents a graphical representation of the fluid velocity for the exact solution.



**Figure 4.1 Velocity distribution of flow at a stagnation point (Reprinted from Schlichting(25:100)).**

In order to allow for direct comparison with Ng's solution, I have utilized his same trial functions as follows:

$$\begin{aligned}\phi_1(x) &= -1 + e^{-x} + x, \\ \phi_i(x) &= (i-1) - ie^{-x} + e^{-ix} \quad i = 2, 3, \dots\end{aligned}\tag{4-3}$$

The special construction of these trial functions is an example of where the homogeneous boundary conditions are automatically satisfied by the approximate solution.

**Solution and Analysis.** Table 4.1 summarizes the quantitative measures of effectiveness for the various solutions as well as for Ng's solution. It can be seen from this table that my results compare very favorably with those reported by Ng. While my solutions tended to produce "better" residuals, he achieved a better value of  $f''(0)$  as compared to the value determined by Froessling. Additionally, comparison of the graph



of my solution for the ten evenly spaced collocation point model (Figure 4.2), with the graph presented in Schlichting (Figure 4.1) shows that my results yield a velocity profile that is very similar to the axially symmetric case.

# collocation points	Spacing	$l_1$ norm resid. (10 pts)	$l_2$ norm resid. (16 pts) <sup>1</sup>	$f''(0)$	Comments
10	even	0.1183	0.0902	1.397	
20	even	0.1445	0.0883	1.403	
100	even	0.1524	0.0868	1.402	
10	uneven	0.1771	0.10506	1.408	
16	even	8.15	3.309	0.556	Ng Actual <sup>2</sup>
16	even	unknown	0.0971	1.313	Ng Reported

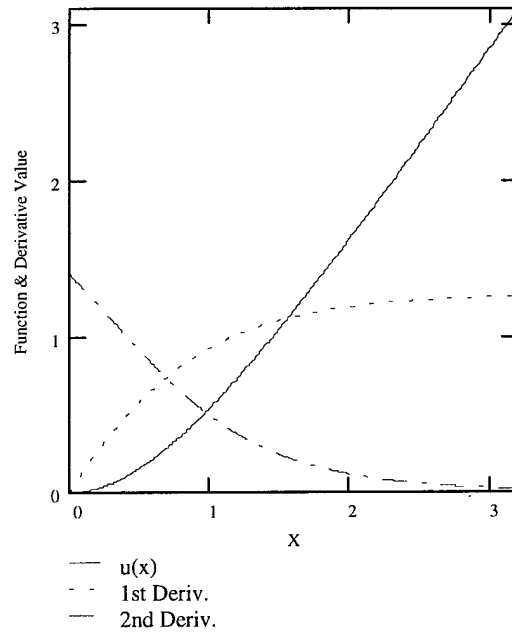
**Table 4.1 Quantitative results for the Navier Stokes solutions.**

Each of my models, as well as Ng's, required the use of four trial functions in the test function. With the exception of Ng's results, the calculated values for the unknown coefficients of the trial functions were very similar to one another. These values and those determined by Ng are summarized in Table 4.2.

---

<sup>1</sup> Although I utilized the  $l_1$  norm over ten points as the optimization metric, Ng utilized the  $l_2$  norm over sixteen points. For comparison purposes both values have been reported in this table.

<sup>2</sup> There is a discrepancy between Ng's reported results and my verification of his calculations. Therefore, I am reporting his results as I calculated them and as he reported them.



**Figure 4.2 Velocity distribution of Navier Stokes solution using ten evenly spaced collocation point model.**

Model:	$c_1$	$c_2$	$c_3$	$c_4$
10 (even)	1.2721	1.0570	-0.5536	0.1112
20 (even)	1.2800	1.0412	-0.5501	0.1120
100 (even)	1.2701	1.0465	-0.5509	0.1120
10 (uneven)	1.2489	1.0681	-0.5407	0.1056
Ng	1.1062	-0.199996	-0.224996	0.099997

**Table 4.2 Coefficients of the trial functions for the Navier Stokes solutions.**

## Linear Boundary Value Problem

**Problem Description.** A scheme designed to approximate the solution of nonlinear boundary value problems should at least be able to accurately approximate

solutions of linear problems. To demonstrate that this is indeed true for this solution technique, I also solved the following linear differential equation

$$f'' + \pi^2 f = -(x + 1) \quad (4-4)$$

on the interval  $(-1, 1)$  with boundary conditions

$$f(-1) = f(1) = 0. \quad (4-5)$$

The exact solution to this problem is as follows:

$$f(x) = -\frac{1}{\pi^3} \sin(\pi x) - \frac{1}{\pi^2} (x + 1).$$

For the test function, I used a combination of Chebyshev polynomials which are orthogonal on the interval  $(-1, 1)$ . The form of the  $n^{th}$  trial function is as follows:

$$\phi_n(x) = T_n(x) - n^2 T_1(x) - (n^2 + (-1)^n) T_0(x) \quad (4-6)$$

where

$$T_n(x) \equiv \cos(n \cos^{-1}(x))$$

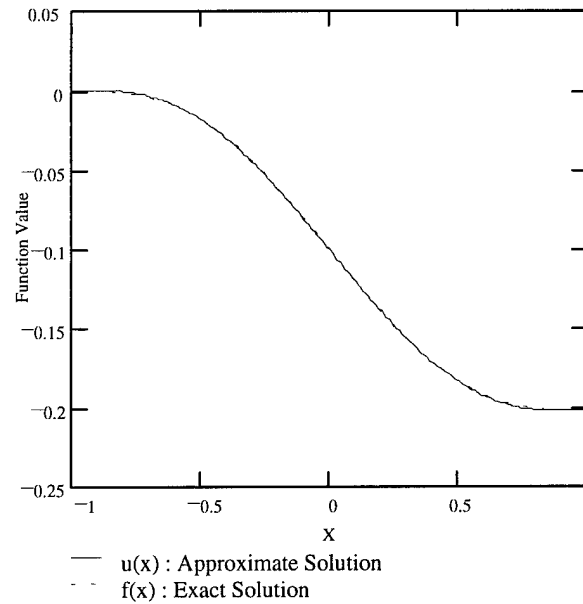
is the  $n^{th}$  degree Chebyshev polynomial. This peculiar formulation is another example where the test function has been specially constructed so that the homogeneous boundary conditions are automatically satisfied.

**Solution and Analysis.** As expected, the modified compromise programming solution produced excellent results for this problem. For each of the models, a four term test function provided the best solution. Table 4.3 summarizes the quantitative solution results. Note that since an exact solution,  $f(x)$ , is available for this problem the  $l_1$  norm of an error vector over ten evenly spaced points is also presented.

# of collocation points	Spacing	$l_1$ norm of residual (10 points)	$l_1$ norm of error (10 points)
10	even	0.1391	0.0138
20	even	0.1982	0.0103
100	even	0.2537	0.0036
10	uneven	0.1437	0.0143

**Table 4.3 Quantitative results of linear boundary value problem.**

While each of the solutions is very good, the solution using the 100 collocation point model is clearly the best. A graph of this solution compared to the exact solution is presented as Figure 4.3. The two solutions are so similar that their graphs cannot be readily distinguished from one another. Table 4.3 clearly shows an improvement in the error as more collocation points are used in the model. It is interesting to note that the same cannot be said of the residual values. While the 100 collocation point model had the smallest error, it did not have the lowest residual. A discussion of this seemingly contradictory result is presented in Chapter Five. Table 4.4 provides the coefficients for the solutions of the various models.



**Figure 4.3 Comparison of exact solution of the linear boundary value problem with solution from the 100 point model.**

Model	$c_1$	$c_2$	$c_3$	$c_4$
10 even	-0.00017	0.01981	0.00003	-0.00237
20 even	-0.00106	0.02047	0.00035	-0.00262
100 even	0.0001	0.02162	-0.00006	-0.00296
10 uneven	-0.00096	0.01975	0.0000755	-0.00239

**Table 4.4 Coefficients of the trial functions for the linear boundary value problem.**

## The Steady Burgers Equation

**Problem Description.** The Burgers equation which models advection and diffusion is a well studied problem in fluid dynamics. Its nonlinearity makes it an ideal equation for the study of turbulence and shock-waves (22:13). The one-dimensional form, which governs the advection and diffusion of one colored liquid in another, is as follows:

$$\left(f - \frac{1}{2}\right) \cdot f' - v \cdot f'' = 0 \quad v > 0 \quad (4-7)$$

on the interval  $(-\infty, \infty)$ . The exact solution to this equation is the following

$$f(x) = \frac{1}{2} \left[ 1 - \tanh\left(\frac{x}{4 \cdot v}\right) \right]. \quad (4-8)$$

While there are no specific boundary conditions on this system, an asymptotic analysis of the exact solution yields the following

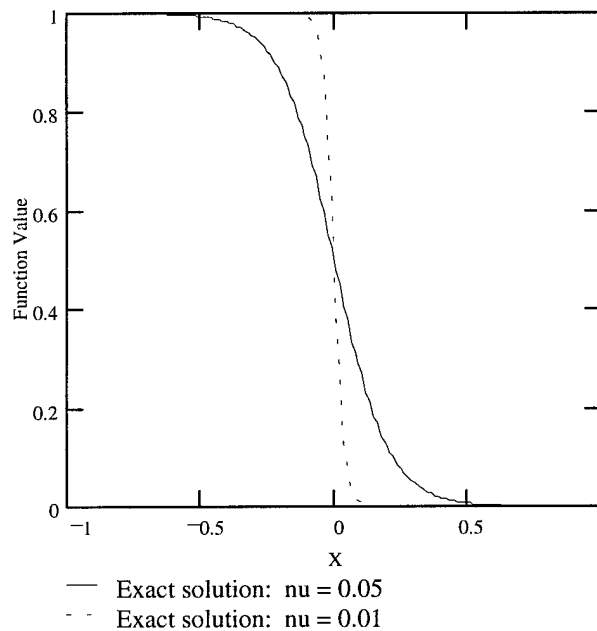
$$\lim_{x \rightarrow -\infty} f(x) = 1 \quad \text{and} \quad \lim_{x \rightarrow \infty} f(x) = 0.$$

Additionally, it is obvious from the exact solution that  $f(0) = 0.5$ . In order to avoid converging to a trivial solution ( $f(x) = 0$ ) to the differential equation, Equation 4-7, the asymptotic limits need to be enforced as boundary conditions. Having access to such information *a priori* is not an unreasonable assumption as the engineer in need of the solution is likely to be keenly aware of such information about the system under study. Hence, the steady Burgers equation can be modeled as a boundary value problem.

To devise an appropriate test function for the solution, *a priori* knowledge is again useful. Here, it is important to realize, that as  $v$  approaches zero, the equation is modeling a shock-like structure imparted on the physical system as can be seen in Figure 4.4. With this information as well as some knowledge about the “boundary” conditions, I chose the following test function:

$$v(x) = c_1 (c_2 - \arctan(c_3 x)) \quad (4-9)$$

With this particular problem I decided to disregard the “modular” approach as described in the methodology whereby successive terms are added to the test function for improvement. Instead I chose to solve the problem with a simple test function as shown in Equation 4-9. Unlike the previous problems this test function does not automatically satisfy the boundary conditions. Therefore, I implemented weighted boundary residuals into the objective function to try to enforce those conditions. For each of the models, experimentation showed that a weighting factor of 25 was sufficient to satisfy these conditions to a reasonable degree (usually out to the fifth decimal place).



**Figure 4.4** Graph of exact solution to the Burgers equation for two values of  $\nu$ .

**Solution and Analysis.** The quality of the solutions of the Burgers equation was mixed -- depending on the model utilized. Table 4.5 summarizes the quality of the results. Because an exact solution was available, the  $l_1$  norm of the error vector is presented. Additionally, I have also included the  $l_1$  norm of the residual and the error

vectors over 100 evenly spaced points. These statistics shall prove useful in the interpretation of the results.

# collocation points	Spacing	$l_1$ norm resid. (10 points)	$l_1$ norm resid. (100 points)	$l_1$ norm error (10 points)	$l_1$ norm error (100 points)
10	even	0.2397	47.3365	0.5041	4.8143
20	even	0.2397	47.3363	0.5041	4.8143
100	even	0.9853	15.6873	0.2522	2.1210
10	uneven	0.4822	4.7590	0.5706	5.5304

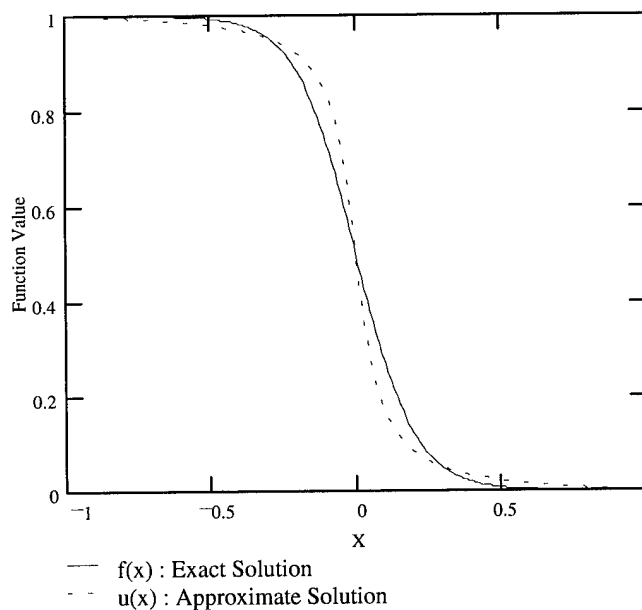
**Table 4.5 Quantitative results for the solutions to the Burgers equation.**

Several important observation can be made from these results. First, the quality of the solution for the 100 point model was much better than that of the other models (See Figure 4.5 and Figure 4.6). In fact, there was only a small difference in the results for the ten and twenty evenly spaced point models. When comparing the residual metric over ten points, one might conclude that the 100 point model produced inferior results. However, when comparing the same metric as well as the error metric over 100 points, this model stands out as the best. One can conclude that by minimizing over significantly more points, the effect of the shock was better captured.

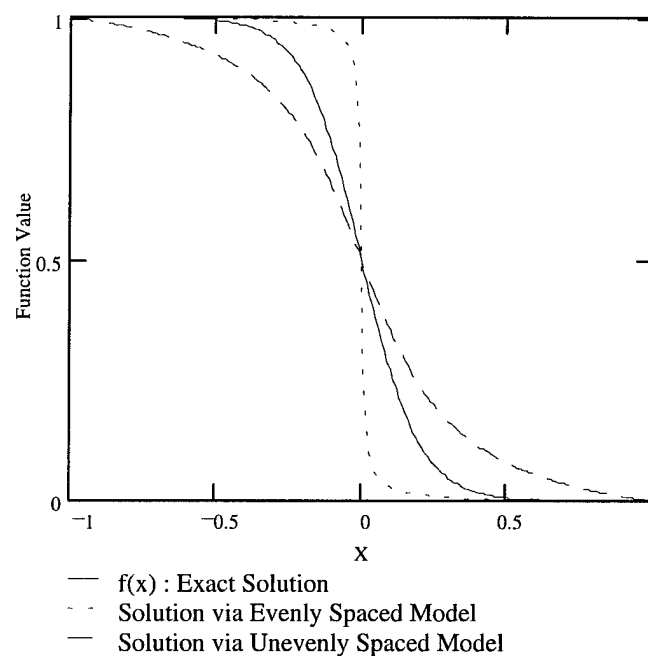
Another significant observation can be made in regard to the unevenly spaced model. Here, I chose to cluster the majority of collocation points about the origin as this seemed to be the critical part of the domain where the most change was occurring. Doing so did not actually improve the solution in comparison to the ten evenly spaced model when looking at the metrics over ten points. However, the uneven spaced model had a significantly lower residual taken over 100 points. One might conclude that clustering the points as such forced a solution that could better take into account the rapid change



that occurred in this area. As an illustration, consider the values of the residuals at a point near the origin,  $x$  equal to 0.01 (which was a collocation point in the unevenly spaced model but not the other). Due to the large magnitude of the first and second derivatives in the evenly spaced model solution at this point, the residual would have been 76.1. However, since the evenly spaced model did not use collocation points so near the origin, the large magnitude of these values was not taken into account in the minimization process. Hence, we have the very sharply descending curve as shown in Figure 4.6. This graph shows a comparison of the exact solution to the solutions generated by evenly and unevenly spaced ten point models. Since  $x = 0.01$  was a collocation point in the unevenly spaced model, a much better representation of the functional characteristics near the origin is achieved as can be seen in the figure. Although one cannot claim that either one of the solutions is better than the other, it can be said that the solution via the unevenly spaced model better reflects the physical properties of the system in the vicinity of the origin which may be the area of most interest to the engineer.



**Figure 4.5 Comparison of the Burgers equation exact solution with approximate solution from 100 point model.**



**Figure 4.6 Graphical comparison of the Burgers equation exact solution to the solutions obtained via the evenly and unevenly spaced ten point models.**

Model:	$c_1$	$c_2$	$c_3$
10 even	0.3204	1.5607	100.0000
20 even	0.3204	1.5608	99.9977
100 even	-0.3317	-1.5073	-15.7101
10 uneven	-0.3745	-1.3351	-4.1645

**Table 4.6 Coefficients of the trial functions for the solutions to the steady Burger equation.**

## The Steady Fisher Equation

**Problem Description.** The Fisher equation has been used to model various phenomena such as population growth with dispersal, the propagation of a virile mutant in an infinitely long environment, and the neutron population in a nuclear reaction (16:581). The form of the time independent ordinary differential equation is given as follows:

$$f''(x) + \lambda f(x)(1 - f(x)) = 0 \quad \lambda \geq 0 \quad (4-10)$$

with boundary conditions

$$f(0) = 0 \quad \text{and} \quad f'(0) = -\frac{1}{2}$$

This differential equation corresponds to a nonlinear oscillation which conserves total energy given by

$$E = \frac{1}{2} f'(x)^2 + \lambda \left( \frac{1}{2} f(x)^2 - \frac{1}{3} f(x)^3 \right) \quad (4-11)$$

Due to the conservation of energy in the system, we now have another measure of the quality of the approximate solutions -- that is, the extent to which energy is constant or conserved in the approximate solutions. Although no exact solution to Equation 4-10 is available, Mickens developed a "best" finite difference scheme to model the solution. This scheme will also be used to compare solutions.

For the test function I used a set of Laguerre functions which are orthogonal on the interval  $(0, \infty)$  defined by

$$L(i, x) \equiv \frac{e^x}{i!} \cdot \frac{d^i}{dx^i} (x^i e^x)$$

Hence, the general form of the test function is

$$v(c_i, x) = \sum_{i=0}^N c_i \cdot L(i, x) \quad (4-12)$$

**Solution and Analysis.** Table 4.7 shows the quantitative description of the results. In the absence of an exact solution, I have used Mickens' finite difference solution as the benchmark to calculate an approximate error norm. Again, there is a wide disparity among the values of the metrics for the various solutions. For each of the solutions the boundary conditions were satisfied to four or five decimal point accuracy. Some similarities can be drawn from these results compared to some of the results for the other problems. For example, from Table 4.7 and Figure 4.7 we again see that the best solutions do not always correspond to the solution with the smallest residual norm. In fact the ten evenly spaced point model yielded a very good approximate function yet had the highest residual norm.

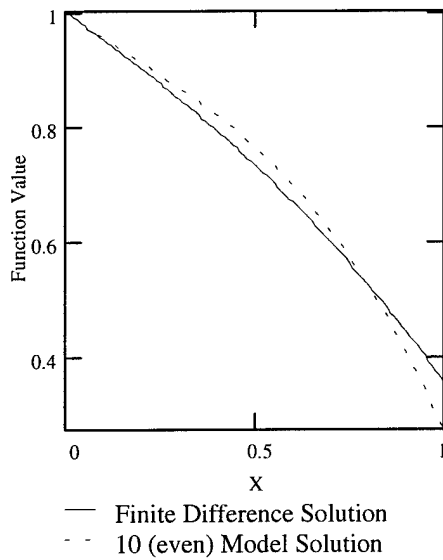
It is also interesting to note that in three of the four models, the residual norm was an order of magnitude larger than the error norm measured over the same points. This observation could also be made with the linear problem already presented. This peculiarity seems more pronounced in the Fisher results since the respective residual norms are so large. Hence, it seems that Ng's desire to yield very small residuals may have been an overly restrictive objective.

# collocation points	Spacing	$l_1$ norm residual (10 points)	$l_1$ norm error (10 points)
10	even	5.1370	0.1767
20	even	2.7820	0.2984
100	even	5.0046	0.4779
10	uneven	4.7314	1.1294

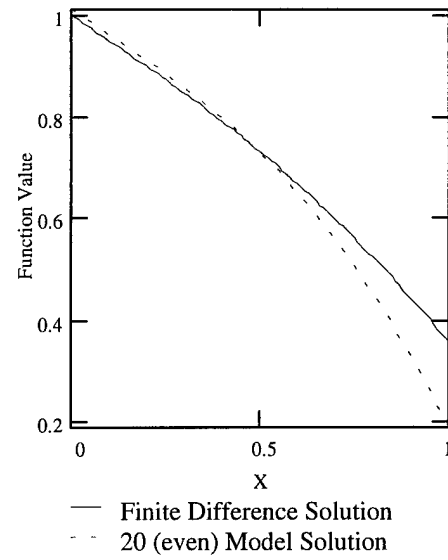
**Table 4.7 Quantitative results for the solutions to the Fisher equation.**

A few observations can be made in regard to the graphs of Figure 4.7. First, minimizing over 100 points did not improve the quality of the solution. This solution was poor in terms of both the residual and the error. Since the underlying function is fairly smooth we would not expect great improvement in the solution by minimizing over many more points. However, we would also not expect such a profoundly worse approximation. My only conjecture as to the cause of this result is that the weighting scheme on the boundary residuals could have used some further adjustment.

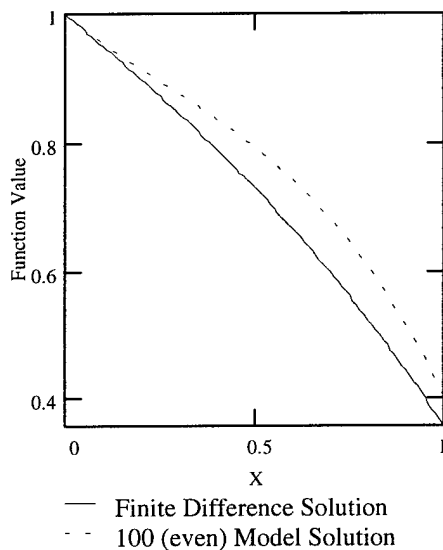
A similar observation can be made for the unevenly spaced model. For the result shown in Figure 4.7, I concentrated the collocation points in the vicinity of  $x = 1$  since this is where the approximating functions in previous solutions were weakest. However, this scheme actually made the solution worse. I also tried other location strategies, such as near the origin: each strategy had results worse than the evenly spaced solution. Additionally, I attempted to weight the collocation residuals in the vicinity of  $x = 1$  -- again with no improvement.



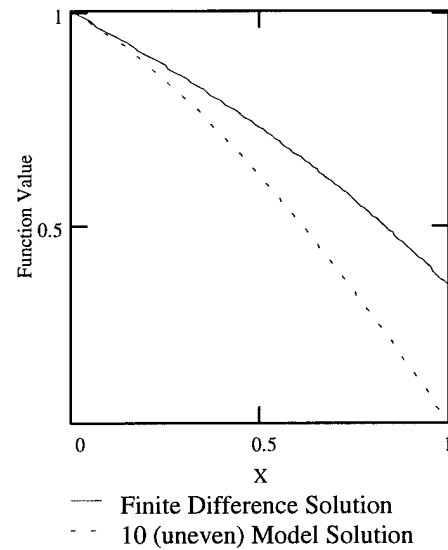
(a)



(b)



(c)

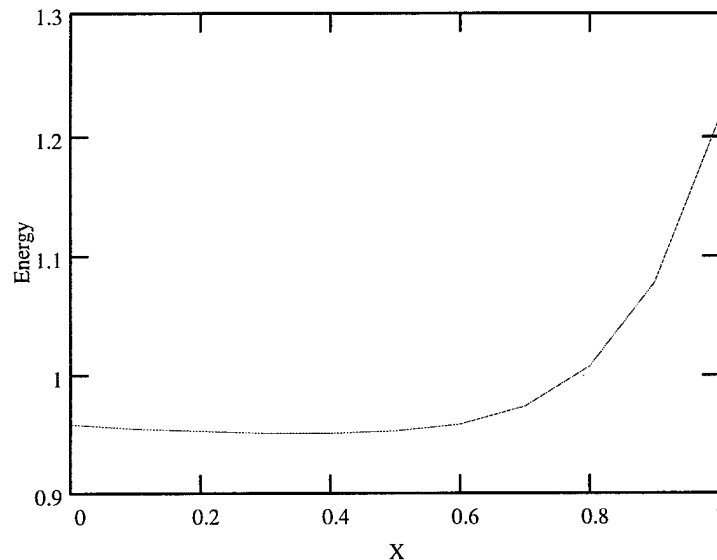


(d)

**Figure 4.7 Graphical comparison of the Fisher equation solutions via (a) 10 pt. (even) ; (b) 20 pt. ; (c) 100 pt. ; and (d) 10 pt. (uneven) models.**

Results for conserved energy property were also mixed. Again, the best results were obtained from the ten and twenty point evenly spaced models. Figure 4.8 represents the energy function on the interval of study for the twenty point model. This graph exhibits a fairly constant function over the first half of the interval, with a gradual increase toward

the end. This characteristic is fairly consistent with finite difference approximations of initial value problems as the error grows as the number of steps taken. Note that the scale of the graph makes the increase appear more pronounced than it actually is.



**Figure 4.8 Graph of energy function for the 20 point model (Fisher equation).**

The trial function coefficients for the Fisher solutions are presented in Table 4.8. For each model, the best solution involved a five term test function. Note, that the indexing of the coefficients starts with zero as shown in Equation 4-12.

Model:	$c_0$	$c_1$	$c_2$	$c_3$	$c_4$
10 even	-1.0299	4.1323	-1.0222	-2.7328	1.6526
20 even	-1.1160	4.7322	-2.8004	-0.5854	0.7813
100 even	-1.0829	5.0367	-3.1831	-0.9126	1.1418
10 uneven	0.1640	0.1985	1.7272	-1.2057	0.1161

**Table 4.8 Coefficients of the trial functions for solutions to the Fisher equation.**

## **V. Observations and Recommendations**

### **Observations**

**Validation of Ng's Work.** The results of my research provide independent validation of the general problem solving framework developed by Ng. As suggested by Ng, the overriding advantage is simplicity by eliminating the need to solve simultaneous nonlinear equations. This technique is likely to be just as useful in other applications. As stated in Chapter One, the method avoids some of the complications inherent to other techniques and is easily adaptable to many classes of problems. In addition the method provides a unique multi-disciplinary approach to complicated problems.

**Modified Compromise Programming.** This research was also successful in demonstrating the utility of the modified compromise programming model. The results indicate that the modification of the generalized compromise programming model is a legitimate transformation which produces very good solutions. More importantly it does so by simplifying the model without sacrificing the quality of the solution. Such a modification is equally suitable for any optimization algorithm regardless of whether or not the algorithm accepts explicit constraints. Hence, the model could be implemented within virtually any nonlinear optimization approach such as simulated annealing, another evolutionary algorithm that requires special handling of constraints.

**Genetic Algorithms.** Because of the high quality of solutions obtained, this research provides further verification of the utility of using genetic algorithms in nonlinear optimization problems. Although I did not implement the model on any other optimization algorithms, the quality of the various solutions indicates that the genetic



algorithm is a very robust global optimizing technique. It would be interesting to compare the solution quality with that of other more traditional algorithms. As a side "experiment" I solved Burgers equation with a test function that included the hyperbolic tangent which was in the exact solution. Within the decimal accuracy inherent to the algorithm and computer, the algorithm quickly converged to the exact solution. This experiment provides further validation of the compromise programming technique and the use of genetic algorithms as a whole.

This study also allows for an observation regarding the stochastic nature of the genetic algorithm. By solving each problem using the three different parameter settings on the algorithm mentioned in Chapter Three, I found there was no uniformly best setting. While for each setting the algorithm tended to converge to its optimal solution in roughly the same amount of generation evaluations, the solutions were often of varying quality. At times these differences were significant. For example, in one stage of computer runs for the Navier - Stokes problem, the residual obtained using Setting 1 was 0.104 while the residual achieved for Setting 2 was 0.766. Such instances not only demonstrate the effects of chaos theory in action but also bring attention to the usefulness of varying the parameter settings. Doing so seems to add some vitality to the solution process.

The last observation with regard to genetic algorithms is simply that the parameter settings I utilized appear to have been appropriate. Early in my research process, I had experimented with settings and often ran the model with much higher population sizes -- typically 100 or 200. I also used varying rates for cross-over and mutation. Due to the much higher population sizes these runs took much more computational time to complete. However, the solutions obtained were no better than those obtained via the three parameter settings that I subsequently standardized. Hence, for this set of problems there seemed to be no need to use large populations.

**Weighted Boundary Residuals.** In solving the steady Fisher and Burgers equations I had to implement weighted boundary residuals in the objective function as individual penalty functions. In contrast, Ng had used lexicographic or preemptive goal programming whereby he treated boundary conditions as explicit constraints. Based on my results, which showed that the boundary conditions were satisfied to a high degree of accuracy, it appears that the weighted residual approach served its purpose. In all instances, a weight ranging from ten to twenty-five on each boundary residual was sufficient. This result is not exactly what I had expected. I had assumed that the 100 point models would require a larger weighting (i.e. on the order of 100) to compensate for the fact that the model minimized over a much larger number of collocation points. However, when utilizing such high weights the algorithm appeared to sacrifice satisfaction of the collocation point residuals in order to minimize the effect of the weighting on the boundary points. Hence, the boundary conditions would be easily satisfied but only at the expense of the other residuals. On the other hand, small weights on the boundary residuals (such as unity) were not large enough to "force" reasonable satisfaction of the boundary conditions.

**Collocation Theory.** The general theory behind collocation methods for linear differential equations is well documented. In general, collocation is the process of interpolating the image of the solution under the differential operator. We assume that if we approximate the image well in the range space of the operator, then the solution will be approximated well also. For certain linear operators and subspaces, collocation solutions exist and are unique. In such cases we can get an estimate of the error,  $\|f(x) - u(x)\|$  (20:274). However, I could find no such documentation for nonlinear boundary value problems. While normal collocation has been successfully used to solve nonlinear problems, it is done more as a "matter of faith" based upon experiences with linear

problems as opposed to explicit theoretical estimates. Likewise, I could find no relevant theory for the application of compromise programming for nonlinear differential equations. However, my research in conjunction with that of Ng indicates that, in the absence of well grounded theory in this area, there is at least practical "field" application of the concept.

However, even though successful application of the procedure has been demonstrated, the results of this study indicate that caution must be used when implementing it. This research has documented several instances where a model that produced the smallest residual did not necessarily yield the best solution. Additionally, those models with seemingly large residuals sometimes produced very good results in terms of error. Hence, it seems that one should use this new technique only when there is strong insight into the expected solution. Otherwise, good solutions could be discarded, whereas inferior solutions might be accepted.

**Choice of Collocation Points** The results of this research indicate that the strategic choice and spacing of collocation points can be critical in obtaining a good solution. Here, I refer to the linear boundary value problem and the Burgers equation. In such problems where a shock is present, we can expect that a concentration of points in the vicinity of the shock should improve the quality of the solution or at least the interpolation. For example, Prenter presented a simple Lagrangian interpolation problem using even spacing. However, this distribution of collocation points produced a solution with very high error at the endpoints. An interpolation using points concentrated around the endpoints would have produced a superior solution (20:38-39). Yet, as noted in the Fisher example, a concentration of collocation points in an area of concern yielded a drastically worse solution than had already been obtained.

**Choice of Test Functions.** Although very little discussion about the proper choice of test functions has been presented, this matter proved to be an important consideration. Naturally, knowledge about the physical system being modeled can only help in the proper selection of a test function. For example, on an infinite domain one would probably not use polynomials to represent a function that was known to “level off” at a certain point. This is because a polynomial will always tend toward positive or negative infinity at the extremities.

The Burgers example was a situation where the test function proved crucial. For my test function, I had initially chosen a combination of Chebyshev polynomials that were very similar to those used for the linear boundary value problem. I did this out of consideration of trying to satisfy the homogeneous “boundary” conditions. However, this function produced some very poor results and was, therefore, abandoned for a test function that I believed could better model the shock that was being modeled. The result was a much better approximation. It is interesting to recall the earlier reference in this chapter where I used a test function that closely resembled what I knew to be the exact solution for the Burgers equation. In this experiment, the procedure quickly converged to the exact solution. Hence, it is clear that a skillfully chosen test function has great impact on the quality of the solution.

## **Recommendations for Further Study**

**Choice of Collocation Points.** While some conclusions about the location and number of collocation points have been made, there is room for more research in this area. At times the results were somewhat counter-intuitive and, hence, might benefit from further study. Each of the examples in this study was done over a relatively small domain even though there may have been some interest in the solution over much larger

intervals. Does expanding the domain necessarily require the addition of more collocation points? It seems clear that the number and location of collocation points needed for a good solution is problem dependent. However, analysis of a number of problems might provide some guidelines in this area. For any study where such specific relationships are trying to be determined, I would recommend the use of a more deterministic algorithm that is not subject to randomness inherent to the genetic algorithm.

**Error Analysis.** While the literature is full of error analysis for linear collocation, the same cannot be said of nonlinear collocation. Hence, it might be that certain nonlinear problems are not suited for collocation. Numerical analysis in this area might prove helpful and prevent a waste of effort in field applications. Such a study would require much mathematical rigor is not likely suited for a master's level thesis.

**Termination Criteria.** As mentioned in Chapter Three, I utilized a somewhat modular approach in building the test functions. Whenever significant reduction in the norm of a residual could be achieved, another trial function would be added. However, this method raises some interesting issues. For example, we have seen that the best solution did not always correspond to the solution with the smallest residual. It is quite likely that I discarded some very good solutions based on the criteria I utilized. Additionally, it would be beneficial to be able to set guidelines that would cause the algorithm to terminate when it could no longer significantly reduce the amount of explanatory error. Such an approach is utilized in step-wise regression where it work very well. However, since very little has been published about error analysis of nonlinear collocation, this recommendation is not likely to be tackled successfully without already having conquered the error issue.

**Other Classes of Problems.** This study did not investigate multi-dimensional differential equations. While those problems studied do have two- and three-dimensional variants, I only worked on the one-dimensional equivalents. Since such a simplification may not always be desired, this research could benefit from experimentation with higher dimensional problems. Additionally, this study only probed the time independent forms of otherwise non-steady partial differential equations. Hence, it seems a natural extension to modify the technique to accommodate non-steady problems. Such studies might be appropriate for a graduate level research effort.

## **Conclusion**

This research investigated a new multi-disciplinary technique of solving engineering problems with notable results. Because this work called on the skills of several disciplines, the successful application of this technique in the field would greatly benefit from a team of operators with experience in the respective disciplines: mathematical science, operations research, and engineering. The mathematician's understanding of functional and numerical analysis combined with the engineers keen insight into the physical aspects of the problem will allow the operations researcher to package the process into an efficient problem solving format. The end result is likely to be the successful application of a robust problem solving technique.

## **Appendix A : Genetic Algorithms**

### **Introduction**

The Genetic Algorithm (GA) is a probabilistic algorithm that has received much attention for its robustness as applied to optimization problems. As algorithms go, the genetic algorithm is fairly young; Holland is credited with its development in the early 1970's (11). As the name implies, genetic algorithms are based on the well known principles of natural selection among the plant and animal species. The genetic algorithm treats optimization as a game of "survival of the fittest" wherein the best "species" divide and conquer while those not capable of competing are doomed to extinction. Consequently, genetic algorithms make use of the basic genetic operations of reproduction, crossover, and mutation, albeit in an artificial sense. The key to applying these operations to an optimization problem is through an encoding scheme designed to take advantage of these powerful tools of nature.

### **Genetic Algorithm Basics**

**Population.** The population for the genetic algorithm is a fixed number of individuals that undergo the three genetic operations. For the simple genetic algorithm (SGA), the number of individuals in the population,  $n$ , remains constant so that when some individuals "die" they are replaced by others. Ideally they will be replaced by

individuals better suited to survive. The value of  $n$  is selected by the user and typically ranges from twenty to two hundred.

**Coding.** As in nature, these individuals have an inherent coding scheme that holds all of their genetic information. For example, consider a problem with a population size of four individuals ( $n = 4$ ). Each individual in this example will be comprised of a single gene of equal length. To represent the genetic code, an alphabet must be used. Although the binary alphabet of "0's" and "1's" is the most common, genetic algorithms are not restricted to this coding scheme. Decimal, octal, hexadecimal, etc. are all valid means of encoding the genes. Before continuing with the example, it is important to first explain what information these genes actually hold.

Each gene is actually the coded representation of the value of the decision variables utilized in the optimization problem. The single-gene individual in this example actually represents a single decision variable. For optimization problems of several variables, the individuals in the population can be comprised of string lengths representing a concatenation of the individual genes. For example, suppose that these genes contain just five characters in their genetic code. One such gene might look like this: *00101*.

Knowledge of binary numbers tells us that this gene represents a decision variable with the decimal equivalent: 5. As previously stated, individuals may be comprised of several genes -- each representing a different decision variable. Now consider the following three-gened individual : *001010001011100*. Separating this individual into thirds yields the genes *00101*, *00010*, and *11100* with the decimal equivalents of the decision variables being 5 , 2, and 28 respectively. Naturally, a gene length of five ( $l = 5$ ) characters is very



limited as it can only represent the integers 0 through 31 ( i.e.,  $2^l - 1$ ). Necessarily, most practical problems require gene lengths to be much longer.

As shall be seen, genetic algorithms exploit the similarities in coding of the most highly fit individuals. In general, consecutive decimal numbers will have similar binary codings. For example, the normal binary coding of the numbers 200 and 201 is

1 1 0 0 1 0 0 0

and

1 1 0 0 1 0 0 1

respectively. Note the similarities in the coding of these two numbers. However, now consider the binary coding of two other consecutive decimal numbers, 127 and 128 given by

0 1 1 1 1 1 1 1

and

1 0 0 0 0 0 0 0

respectively. These two codings could not be any more different. This peculiarity is known as a Hamming cliff. Since genetic algorithms have been shown to work best when similarly valued decimal numbers are coded similarly in binary form, another coding known as Gray coding is often used (5:874). Gray coding is a system that ensures that the coding of consecutive integers will differ at only one bit position.

**The Fitness Function.** The fitness function (more commonly known as the objective function in most optimization problems) is the means to evaluate the suitability

of each individual in the population. A relative scaling system is used in order to compare one individual's fitness with the others. As one might expect, those with the highest fitness level have the highest probability of being selected to carry on the three genetic operations that will affect future generations of the population. As shall be seen, this is where the non-deterministic nature of the algorithm takes hold. The probability that an individual is selected to reproduce and carry on the other genetic operations is proportional to the fitness of that individual relative to the population.

## **Genetic Operators**

**Reproduction.** Reproduction, also known as selection, is simply the process of individuals in the population being exactly duplicated to fill the positions in the next generation. The probability that an individual reproduces is proportional to that individual's fitness relative to the population. This selection process is controlled by the spin of a simulated roulette wheel which is "spun"  $n$  times. Hence, it is possible that some individuals may produce multiple offspring (each one identical), while other individuals may not produce any offspring.

In order to maintain each successive generation at a constant population size, as required by the algorithm, no more operations are performed on the "parent" population once they reproduce. These individuals (as well as those not selected to reproduce), in effect, die. All subsequent operations are performed on future generations. However, some modifications to the basic genetic algorithm call for an *elitist strategy* wherein the most fit individual of a generation is always selected to reproduce itself at least once.

This strategy ensures the preservation of the best genetic material so as not to leave the fate of future generations strictly up to chance.

For an illustration of how the roulette wheel analogy is applied, consider the example of a four member population ( $n = 4$ ) for a fitness function that is to be maximized. For this example, I will ignore the actual form of the notional fitness function and accept that the fitness levels as shown in Table A.1 are accurate. This simple example shows that the fitness level (third column) for individual number 1 accounts for one half of the sum of all fitness values, thus giving it a relative fitness (fourth column) of 0.5. In this example the total number of slots on the wheel is equal to the number in the population (although other schemes are possible). Thus, in spinning this wheel four times, one can see that individual number 1 has the highest probability of reproducing itself.

Individual (i)	Genetic Code	Fitness ( $f_i$ )	Relative Fitness ( $r = f_i / n$ )	Number of Slots on Wheel ( $n \times r$ )
1	01110	50	.50	2
2	00010	0	0	0
3	11000	25	.25	1
4	01100	25	.25	1
Total		100	1.00	4

**Table A.1 Four member population example.**

This example only presents the very basic fundamentals of selection. It does not take into account such things as negative fitness values, non-integer values for number of slots, and procedures of selection for minimization problems. Such situations are easily handled by slight modifications of the algorithm that still keep intact the same basic principles.

**Crossover.** Crossover, also known as recombination, is the means by which genes exchange their genetic material with one another. With crossover, there are two probabilistic elements: 1) which genes are selected to cross; and 2) where they cross, once selected. Generally, the probability for each event is fixed and does not vary with fitness level as does reproduction. When two genes are selected to cross at a selected point, they are simply split at that point and recombined with one another to form two new genes of the same length. For example consider the first and third genes in the above example splitting at the third position as follows:

0 1 1 | 1 0

1 1 0 | 0 0

to form the following two new genes:

0 1 1 0 0

1 1 0 1 0

Since crossover is a non-deterministic event, not all genes are selected to perform it. Additionally, the crossover can occur at any position within a gene based on an equal

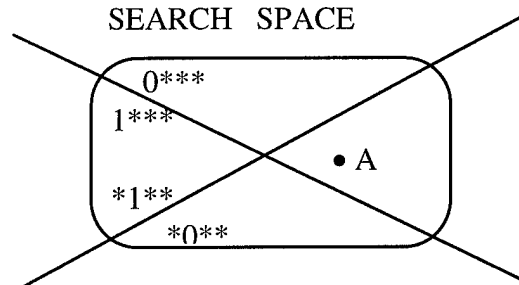
probability for all possible locations. Note also that crossover does not result in a change of the population size,  $n$ . The purpose of crossover is for the most fit individuals (i.e., those who were selected to reproduce) to share their genetic information with other strong performing individuals in hopes of yielding even better individuals. A rough analogy of this process is the "all-pro" defensive end who inherits his size from his father but his quickness from his mother. Hence, it would be advantageous if the pairs selected to recombine did so in a manner such that their "best" characteristics were shared.

**Mutation.** Mutation is the process by which individual positions within a gene change to a different character at random. Thus, for a binary coding, a "1" would mutate to a "0" and vice versa. For example if the gene "0 1 1 0 0" mutates at its fifth position, it would become "0 1 1 0 1." Since, mutation is not the most powerful operation occurring in nature, the probability of a bit mutating in genetic algorithms is normally chosen to be small -- typically on the order of 0.001. Although not the most powerful operator, mutation does maintain vitality in the system by regaining lost genetic material. For example, it is possible for all of the individuals in a population to converge to a specific bit, such as a "1," in a given position. If the optimal solution contains a "0" in that position, then mutation is the only way for a "0" to be restored (27:19).

## Schema Theory

The power behind genetic algorithms is based on schema theory. A schema is a similarity template of the subsets of gene strings. To explain scheme theory, a new character, " \* ", must be introduced. Similar to a wild card as used in computer file names, this symbol is essentially a "don't care" symbol. For binary coding it could represent either a "0" or a "1." For example, the schema 0 1 \* 0 could represent either

0110 or 0100. Such schema define hyperplanes in the search space as Figure A.1 illustrates (5:875).



**Figure A.1 Illustration of sample schema and defining hyperplanes.**

This figure shows four hyperplanes established by four schema (although more are possible). Note that the point “A” lies in exactly two of the hyperplanes and thus, is a member of only two of these schema. The fitness of a given individual actually provides some information about the average fitness of the  $2^l$  different schema of which it is an instance (5:876). Since in each generation, the fitness of  $n$  individuals is evaluated, the effect of this observation is compounded. A general result of the study of schema theory by Holland is that in each generation of size  $n$ , the genetic algorithm will implicitly evaluate  $n^3$  schemata (27:21). By combining this observation with the results of Holland’s “fundamental theorem of genetic algorithms” which states that the best schema are expected to grow exponentially with time, one can see that genetic algorithms do a very efficient job of searching the solution space (27:21). This implicit parallelism is what gives the genetic algorithm its power.

Just as in nature, genetic algorithms do not realize terrific gains until very many generations. It took millions of years for life forms as we know them to evolve into their present states via the workings of genetics. Even though no optimization problem can

rival the complexity of human evolution, it may still take thousands of generations for a solution to completely "evolve." Yet the power of modern computing resources acting on the simulated genetic operators of reproduction, cross-over, and mutation has proven very effective in solving very complicated optimization problems.

## Appendix B: Sample Computer Code

### General

This appendix provides sample computer code utilized for the genetic algorithm software, Genesis, version 5.0. Genesis is written in the C programming language and requires the user to provide the appropriate input file containing the achievement function which is to be optimized. The input file is compiled and linked to the remainder of the computer code via a makefile that is included with the software package. The user may vary the parameters of the algorithm (i.e., population size, cross-over rate, etc.) via a setup program provided with the software.

The input files for the models using ten evenly spaced collocation points is provided for each of the four problems solved. For the models using other distributions of collocation points, only the function that sums residuals need be changed.

### Input File for the Navier - Stokes Problem

```

/***** file ns10e.c *****/

/* This file creates the achievement function for the following differential */
/* equation:  $f''' + 2ff'' + 1 - f^2 = 0$  on (0,1.5) */
/* The boundary conditions are  $f(0)=f'(0)=0$  and  $\lim_{x \rightarrow \infty} f(x) = 1$  */
/* The trial function is a combination of exponential functions */
/* It used 10 evenly spaced collocation pts from 0 to 1.35 */

#include "math.h"

/***** FUNCTION TO SUM RESIDUALS *****/

double eval(str, length, vect, genes)
char str[]; /* string representation */
int length; /* length of bit string */
double vect[]; /* floating point representation */
int genes; /* number of elements in vect */
{
double resid(), u(), du(), ddu(), dddu(), eval();
int m; /* number of collocation points */

```



```

double l1norm ;    /* l1 norm of residuals          */
int j;             /* counter                                          */
double x[10];

m = 10;
l1norm = 0.0;

for (j = 0; j < m ; j++)
{
    x[j] = j * 0.15 ;          /* define collocation points */
    l1norm += fabs( resid( vect, x[j], genes) );
}
return (l1norm);
}
/* *****END FUNCTION*****/

/* FUNCTION TO CALCULATE RESIDUAL AT COLLOCATION POINT */

double resid( vect, x, genes)
double vect[];
double x;
int genes;
{
    double residual;

    residual = dddu(vect, x, genes) + 2.0 * u(vect, x, genes) * ddu(vect, x, genes)
              + 1.0 - pow( du(vect, x, genes),2.0);

    return (residual);
}
/* *****END FUNCTION*****/

/***** FUNCTION TO GENERATE TEST FUNCTION *****/

double u( vect, x, genes)

double vect[];
double x;
int genes;
{
    double test_function = 0.0;
    int i;

```

```

for (i = 2 ; i <= genes; i++ )

    test_function += vect[i-1] * ((i-1) - i * exp(-x) + exp(-i * x));

test_function = test_function + vect[0] * (-1.0 + exp(-x) + x);
return (test_function);
}
/* *****END FUNCTION*****/

/* FUNCTION TO GENERATE FIRST DERIVATIVE OF TEST FUNCTION */

double du( vect, x, genes)
    double vect[];
    double x;
    int genes;
{
    double first_derivative = 0.0;
    int i;

    for (i = 2 ; i <= genes; i++ )

        first_derivative += vect[i-1] * (i * exp(-x) - i * exp(-i * x));

    first_derivative = first_derivative + vect[0] * ( -exp(-x) + 1.0);

    return (first_derivative);
}
/* *****END FUNCTION*****/

/* FUNCTION TO GENERATE SECOND DERIVATIVE OF TEST FUNCTION */

double ddu( vect, x, genes)
    double vect[];
    double x;
    int genes;
{
    double second_derivative = 0.0;
    int i;

    for (i = 2 ; i <= genes; i++ )

```

```

    second_derivative += vect[i-1] * (-i * exp(-x) + i*i * exp(-i * x));

second_derivative = second_derivative + vect[0] * exp(-x);

return (second_derivative);
}
/* *****END FUNCTION***** */

/* FUNCTION TO GENERATE THIRD DERIVATIVE OF TEST FUNCTION */

double dddu( vect, x, genes)

    double vect[];
    double x;
    int genes;
{
    double third_derivative = 0.0;
    int i;

    for (i = 2 ; i <= genes; i++ )
    {
        third_derivative += vect[i-1] * (i * exp(-x) - i*i*i * exp(-i * x));
    }
    third_derivative = third_derivative + vect[0] * (-exp(-x));

    return (third_derivative);
}
/* *****END FUNCTION***** */

/****** end of file *****/

```

## Input File for the Linear Boundary Value Problem

```
/****** file ch10e.c *****/
```

```
/* This file is the achievement function for the following differential  
equation:
```

$$f'' + (\pi)^2 * f = -(x + 1) \text{ on the interval } (-1, 1).$$

With the following boundary conditions:

$$f(-1) = 0 \quad \text{and} \quad f(1) = 0$$

over 10 evenly spaced collocation points

The trial functions are a combination of Chebyshev polynomials. \*/

```
#include "math.h"
```

```
/****** FUNCTION TO SUM RESIDUALS *****/
```

```
double eval(str, length, vect, genes)
char str[];      /* string representation */
int length;      /* length of bit string */
double vect[];   /* floating point representation */
int genes;       /* number of elements in vect */
{
    double resid(), u(), ddu(), eval();
    int m;        /* number of collocation points */
    double l1norm; /* sum of residuals */
    int i;        /* counter */
    double x[10];

    m = 10;
    l1norm = 0.0;

    for (i = 0; i < m; ++i)
    {
        x[i] = (i - 4.5)/4.5; /* define collocation points */
        l1norm += fabs( resid( vect, x[i], genes) );
    }

    return (l1norm);
}
```

```

}
/* *****END FUNCTION***** */

/* FUNCTION TO CALCULATE RESIDUAL AT COLLOCATION POINT */

double resid( vect, x, genes)
double vect[];
double x;
int genes;      /* this may need to be double */
{
double residual;
double pi_sqrd = 9.8696044;

residual = ddu(vect, x, genes) + pi_sqrd * u(vect,x,genes) + x + 1;

return (residual);
}
/* *****END FUNCTION***** */

/****** FUNCTION TO GENERATE TEST FUNCTION ***** */

double u( vect, x, genes)

double vect[];
double x;
int genes;
{
double test_function = 0.0;

/* This is four a four term test function */

test_function = vect[0] * (2 * pow(x, 2.0) - 4 * x - 6)
               + vect[1] * (4 * pow(x, 3.0) - 12 * x - 8)
               + vect[2] * (8 * pow(x, 4.0) - 8 * pow(x, 2.0) - 16 * x - 16 )
               + vect[3] * (16 * pow(x, 5.0) - 20 * pow(x, 3.0) - 20 * x - 24);

return (test_function);
}
/* *****END FUNCTION***** */

* FUNCTION TO GENERATE SECOND DERIVATIVE OF TEST FUNCTION */

double ddu( vect, x, genes)

```

```

double vect[];
double x;
int genes;
{
double second_derivative = 0.0;

second_derivative = vect[0] * 4 + vect[1] * 24 * x
                    + vect[2] * (96 * pow(x, 2.0) - 16)
                    + vect[3] * (320 * pow(x, 3.0) - 120*x );

return (second_derivative);
}
/* *****END FUNCTION***** */

/***** end of file *****/

```

## Input File for the Burgers Equation Problem

```
/****** file bu10e.c *****/

/* This file creates the achievement function for the following differential */
/* equation:  $(u - .5)u' - \nu u'' = 0$   $\nu > 0$  on  $(-\infty, \infty)$  */
/* This problem uses asymptotic boundary condtions */
/* The test function is:  $a(b - \arctan(c x))$  */
/* It uses 10 evenly spaced collocation pts from -1 and 1 */

#include "math.h"

/****** FUNCTION TO SUM RESIDUALS *****/

double eval(str, length, vect, genes)
char str[]; /* string representation */
int length; /* length of bit string */
double vect[]; /* floating point representation */
int genes; /* number of elements in vect */
{
double resid(), u(), du(), ddu(), eval();
int m; /* number of collocation points */
double l1norm; /* l1 norm of residuals */
int j; /* counter */
double x[10];

m = 10;
l1norm = 0.0;

for (j = 0; j < m ; j++)
{
x[j] = (j - 4.5)/4.5 ; /* define collocation points */

l1norm += fabs( resid( vect, x[j], genes) )
+ 10 * fabs(1 - u(vect, -1.0, genes) )
+ 10 * fabs(du(vect, 1.0, genes) )
+ 10 * fabs(u(vect, 1.0, genes) );
}
return (l1norm);
}
```

```

/* *****END FUNCTION***** */

/* FUNCTION TO CALCULATE RESIDUAL AT COLLOCATION POINT */

double resid( vect, x, genes)
double vect[];
double x;
int genes;
{
double residual;
double nu = 0.05;

residual = (u(vect, x, genes) - 0.5)* du(vect, x, genes)
           - nu * ddu(vect, x, genes);

return (residual);
}
/* *****END FUNCTION***** */

/***** FUNCTION TO GENERATE TEST FUNCTION *****/

double u( vect, x, genes)

double vect[];
double x;
int genes;
{
double test_function = 0.0;
double Pi = 1.314159;
int i;

test_function = vect[0]* ( vect[1] - atan(vect[2] * x) );

return (test_function);
}
/* *****END FUNCTION***** */

/* FUNCTION TO GENERATE FIRST DERIVATIVE OF TEST FUNCTION */

double du( vect, x, genes)
double vect[];

```



```

double x;
int genes;
{
double first_derivative = 0.0;
double Pi = 1.314159;
int i;

    first_derivative = -vect[2] / ( (1 + vect[2]*vect[2] * x*x) * Pi );

return (first_derivative);
}
/* *****END FUNCTION*****/

/* FUNCTION TO GENERATE SECOND DERIVATIVE OF TEST FUNCTION */

double ddu( vect, x, genes)
double vect[];
double x;
int genes;
{
double second_derivative = 0.0;
double Pi = 1.314159;
int i;

    second_derivative = 2.0 * pow(vect[2], 3.0) * x
                        / (pow(1 + vect[2]*vect[2] * x*x, 2.0) * Pi);

return (second_derivative);
}
/* *****END FUNCTION*****/

/****** end of file *****/

```

## Input File for the Fisher Equation Problem

```
/****** file f10e5.c *****/
```

```
/* This file creates the achievement function for the following differential
equation:
```

$$f' + \lambda f(1 - f) = 0 \text{ on the interval } (0, \infty).$$

With the following boundary conditions:

$$f(0) = 1 \text{ and } f(\infty) = -0.5$$

It uses Laguerre polynomials as the trial functions over 10 evenly spaced collocation points. The boundary conditions are weighted by a factor of 25. \*/

```
#include "math.h"
```

```
/****** FUNCTION TO SUM RESIDUALS *****/
```

```
double eval(str, length, vect, genes)
char str[];          /* string representation */
int length;          /* length of bit string */
double vect[];       /* floating point representation */
int genes;           /* number of elements in vect */
{
double resid(), u(), ddu(), eval();
int m;               /* number of collocation points */
double l1norm;       /* sum of residuals */
double ach_fnc;      /* l1norm of residual eqn plus boundary residuals */
int i;               /* counter */
double x[10];

m = 10;
l1norm = 0.0;
ach_fnc = 0.0;

for (i = 0; i < m; i++)
{
x[i] = i * .11111; /* define collocation points */
l1norm += fabs( resid( vect, x[i], genes) );
}
```

```

    }
    ach_fnc = llnorm + 25 * fabs( 1- u(vect, 0.0,genes) )
               + 25 * fabs(-.5 - du(vect,0.0,genes));

    return (ach_fnc);
}
/* *****END FUNCTION***** */

/* FUNCTION TO CALCULATE RESIDUAL AT COLLOCATION POINT */

double resid( vect, x, genes)
    double vect[];
    double x;
    int genes;
{
    double residual;
    double lambda= 5.0;

    residual = ddu(vect, x, genes) + lambda * u(vect,x,genes) * (1.0 - u(vect,x,genes) ) ;

    return (residual);
}
/* *****END FUNCTION***** */

/****** FUNCTION TO GENERATE TEST FUNCTION ***** */

double u( vect, x, genes)

    double vect[];
    double x;
    int genes;
{
    double test_function = 0.0;

    /* This is four a five term test function */

    test_function = vect[0]
        + vect[1] * ( exp(x)*(exp(-x) - x/exp(x) ) )
        + vect[2] * ( exp(x)*(2.0/exp(x) - 4*x/exp(x)
            + pow(x,2.0)/exp(x))/2.0 )
        + vect[3] * ( exp(x)*(6/exp(x) - 18*x/exp(x)
            + 9*pow(x,2.0)/exp(x)
            - pow(x,3.0)/exp(x))/6 )

```

```

        + vect[4] * ( exp(x)*(24/exp(x) - 96*x/exp(x)
          + 72.0*pow(x,2.0)/exp(x) - 16*pow(x,3.0)/exp(x)
          + pow(x,4.0)/exp(x))/24 );

return (test_function);
}
/* *****END FUNCTION*****/

/* FUNCTION TO GENERATE FIRST DERIVATIVE OF TEST FUNCTION */

double du( vect, x, genes)
double vect[];
double x;
int genes;
{
double first_derivative = 0.0;

first_derivative = vect[1] * ( exp(x)*(exp(-x) - x/exp(x))
          + exp(x)*(-2/exp(x) + x/exp(x)) )
+ vect[2] * ( exp(x)*(-6/exp(x) + 6*x/exp(x)
          - pow(x,2.0)/exp(x))/2
          + exp(x)*(2/exp(x) - 4*x/exp(x)
          + pow(x,2.0)/exp(x))/2
          + vect[3] * ( exp(x)*(6/exp(x) - 18*x/exp(x)
          + 9*pow(x,2.0)/exp(x)
          - pow(x,3.0)/exp(x))/6
          + exp(x)*(-24/exp(x) + 36*x/exp(x)
          - 12*pow(x,2.0)/exp(x)
          + pow(x,3.0)/exp(x))/6
          + vect[4] * ( exp(x)*(-120/exp(x) + 240*x/exp(x)
          - 120*pow(x,2.0)/exp(x)
          + 20*pow(x,3.0)/exp(x)
          - pow(x,4.0)/exp(x))/24
          + exp(x)*(24/exp(x) - 96*x/exp(x)
          + 72*pow(x,2.0)/exp(x)
          - 16*pow(x,3.0)/exp(x)
          + pow(x,4.0)/exp(x))/24 );

return (first_derivative);
}
/* *****END FUNCTION*****/

```

```
/* FUNCTION TO GENERATE SECOND DERIVATIVE OF TEST FUNCTION */
```

```
double ddu( vect, x, genes)
double vect[];
double x;
int genes;
{
double second_derivative = 0.0;

second_derivative = vect[1] * ( exp(x)*(exp(-x) - x/exp(x))
                               + exp(x)*(3/exp(x) - x/exp(x))
                               + 2*exp(x)*(-2/exp(x) + x/exp(x)) )
+ vect[2] * ( exp(x)*(-6/exp(x) + 6*x/exp(x))
              - pow(x,2.0)/exp(x)
              + exp(x)*(12/exp(x) - 8*x/exp(x))
              + pow(x,2.0)/exp(x))/2
              + exp(x)*(2/exp(x) - 4*x/exp(x))
              + pow(x,2.0)/exp(x))/2 )
+ vect[3] * ( exp(x)*(6/exp(x) - 18*x/exp(x))
              + 9*pow(x,2.0)/exp(x)
              - pow(x,3.0)/exp(x))/6
              + exp(x)*(60/exp(x) - 60*x/exp(x))
              + 15*pow(x,2.0)/exp(x)
              - pow(x,3.0)/exp(x))/6
              + exp(x)*(-24/exp(x) + 36*x/exp(x))
              - 12*pow(x,2.0)/exp(x)
              + pow(x,3.0)/exp(x))/3 )
+ vect[4] * ( exp(x)*(-120/exp(x) + 240*x/exp(x))
              - 120*pow(x,2.0)/exp(x)
              + 20*pow(x,3.0)/exp(x)
              - pow(x,4.0)/exp(x))/12
              + exp(x)*(360/exp(x) - 480*x/exp(x))
              + 180*pow(x,2.0)/exp(x)
              - 24*pow(x,3.0)/exp(x)
              + pow(x,4.0)/exp(x))/24
              + exp(x)*(24/exp(x) - 96*x/exp(x))
              + 72*pow(x,2.0)/exp(x)
              - 16*pow(x,3.0)/exp(x)
              + pow(x,4.0)/exp(x))/24 );

return (second_derivative);
}
/* *****END FUNCTION*****/
/***** end of file *****/
```

## References

1. Ascher, Uri and others. *Numerical Solutions of Boundary Value Problems for Ordinary Differential Equations*. Englewood Cliffs New Jersey: Prentice-Hall, 1988.
2. Barrodale, Ian. "L<sub>1</sub> Approximation and the Analysis of Data." *Applied Statistics*, 17: 51-57 (1968).
3. Barrodale, Ian and Andrew Young. "Algorithms for Best L<sub>1</sub> and L<sub>∞</sub> Linear Approximations on a Discrete Set," *Numerische Mathematik*, 8: 295-306. (1966)
4. Bloomfield, Peter and William L. Steiger. *Least Absolute Deviations: Theory, Applications, and Algorithms*. Boston: Birkhauser Boston Inc., 1983.
5. Forrest, Stephanie. "Genetic Algorithms: Principles of Natural Selection Applied to Computation," *Science*, 261: 872 - 878 (13 August 1993).
6. Gates, George H. and others. "Simple Genetic Algorithm Parameter Selection for Protein Structure Prediction," Unpublished paper submitted to the International Conference on Genetic Algorithms. March 1995.
7. Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading Massachusetts: Addison-Wesley, 1989.
8. Grefenstette, John J. "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16, No. 1: 102-106 (January/February 1986).
9. Grefenstette, John J. *Genesis*. Version 5.0, IBM, 80k, disk. Computer software. The Software Partnership, Melrose MA, 1990.
10. Hooke, Robert and T. A. Jeeves. " 'Direct Search' Solution of Numerical and Statistical Problems," *Journal of the Association for Computing Machinery*, 8: 212 - 229 (April 1961).
11. Holland, John. *Adaptation in Natural and Artificial Systems*. Ann Arbor Michigan: University of Michigan, 1975.
12. Ignizio, James P. *Goal Programming and Extensions*. Toronto: D. C. Heath, 1976.

13. Ignizio, James P. "Goal Programming: An Overview," *Computers and Operations Research*, 10: 277-287 (No. 4 1983)
14. Karr, C. L. and others. *Genetic Algorithms Applied to Least Squares Curve Fitting*. United States Department of the Interior Report of Investigations 9339. Washington: Government Printing Office, 1991.
15. McCormick, Garth. *Nonlinear Programming: Theory, Algorithms, and Applications*. New York: Wiley, 1983.
16. Mickens, Ronald E. "A Best Finite-Difference Scheme for the Fisher Equation," *Numerical Methods for Partial Differential Equations*, 10: 581-585 (August 1994).
17. Ng, Kevin Y. "Solution of Navier-Stokes Equations by Goal Programming," *Journal of Computational Physics*, 39: 103-111 (January 1981).
18. Ng, Kevin Y. "Goal Programming, Method of Weighted Residuals, and Optimal Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics, SMC-17, No. 1*: 102-106 (January/February 1987).
19. Powell, Michael J. *Approximation Theory and Methods*. Cambridge: Cambridge University Press, 1981.
20. Prenter, P. M. *Splines and Variational Methods*. New York: Wiley, 1989.
21. Richardson, Jon T. and others. "Some Guidelines for Genetic Algorithms with Penalty Functions," *Proceedings of the Fourth International Conference on Genetic Algorithms*. 191-197. San Mateo, California: Morgan Kaufmann, 1989.
22. Roache, Patrick. *Computational Fluid Dynamics*. Albuquerque: Hermosa Publishers, 1985.
23. Schaffer, J. David "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms," *Proceedings of the First International Conference on Genetic Algorithms*. 93-100. Hillsdale New Jersey: Lawrence Elbaum and Associates, 1985.
24. Schaffer, J. David and others. "A Study of Control Parameters Affecting Online Performance for Genetic Algorithms for Function Optimization," *Proceedings of the Third International Conference on Genetic Algorithms*. 51-60. San Mateo California: Morgan Kaufmann Publishers, 1989.
25. Schlichting, Hermann. *Boundary-Layer Theory*. New York: McGraw-Hill, 1968.

26. Seo, Fumiko and Masatoshi Sakawa. *Multiple Criteria Decision Analysis in Regional Planning*. Boston: D. Reidel Publishing, 1988.
27. Srvinas, M and Lalit M. Patnaik. "Genetic Algorithms: A Survey," *Computer*, 27: 17-26 (June 1994)
28. Sutton, Patrick and Sheri Boyden. "Genetic Algorithms: A General Search Procedure," *American Journal of Physics*, 62: 549-552 (June 1994).
29. Villadsen, John and Michael Michelsen. *Solution of Differential Equations by Polynomial Approximation*. Englewood Cliffs New Jersey: Prentice-Hall, 1966.
30. Zeleny, Milan. *Multiple Criteria Decision Making*. New York: McGraw-Hill, 1982.



## Vita

Captain John L. Zornick was born on 24 April 1963 in Buffalo, New York. He graduated from Salem High School in Salem, Ohio in 1981 and attended the United States Military Academy, graduating with a Bachelor of Science in Engineering Management. Upon graduation in 1985 he was commissioned as a second lieutenant in the regular army as an infantry officer. His first duty assignment was at Fort Carson, Colorado where he served as a platoon leader, executive officer, and battalion maintenance officer in the 4th Infantry Division (Mechanized). He subsequently served with the 7th Infantry Division (Light) at Fort Ord, California where he held positions as a rifle company commander and a long range surveillance unit commander. Captain Zornick is a graduate of the Infantry Officer Basic Course, the Infantry Officer Advanced Course, Airborne School, Jumpmaster School, and Ranger School. He was assigned to attend the Air Force Institute of Technology School of Engineering in 1993 to pursue a M.S. in Operations Research.

Permanent Address:

2307 Kennedy Dr.  
Salem, Ohio 44460

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE A ROBUST METHOD OF SOLVING NONLINEAR BOUNDARY VALUE PROBLEMS VIA MODIFIED COMPROMISE PROGRAMMING		5. FUNDING NUMBERS		
6. AUTHOR(S) John L. Zornick, Captain, USA				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB, OH 45433-6583		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/ENC/GOR/95J-01		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)  N/A		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) This study is an extension of Ng's previous work in which goal programming was used to determine an approximate solution to a boundary value problem. This approach follows the same basic approach developed by Ng in which the method of collocation was recast as a compromise programming model. Hence, instead of solving a system of simultaneous nonlinear equations, one seeks a compromise solution which minimizes, in a weighted residual sense, a vector norm of the differential equation residuals. A difference in this approach is that it uses a genetic algorithm as the optimizing engine as opposed to the pattern search used by Ng. This model also differs in that it eliminates the need for deviation variables. As a result, this model is a simplification in that the number of decision variables is independent of the number of collocation points. This technique is robust in that it has been demonstrated on a variety of problems and has produced good results. The results of four example problems compare favorably with those of Ng and other solution techniques.				
14. SUBJECT TERMS Boundary Value Problems, Nonlinear Differential Equations, Goal Programming, Compromise Programming, Genetic Algorithms			15. NUMBER OF PAGES 89	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	